INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

P.1202.2
(05/2013)

SERIES P: TERMINALS AND SUBJECTIVE AND OBJECTIVE ASSESSMENT METHODS

Models and tools for quality assessment of streamed media

Parametric non-intrusive bitstream assessment of video media streaming quality – higher resolution application area

Recommendation ITU-T P.1202.2

Parametric non-intrusive bitstream assessment of video media streaming quality – higher resolution application area

Summary

Recommendation ITU-T P.1202.2 specifies the algorithmic model for the higher resolution (HR) application area of ITU-T P.1202. The ITU-T P.1202 series of documents specifies models for monitoring the video quality of IP-based video services based on packet-header and bitstream information. The higher resolution application area of ITU-T P.1202.2 part of ITU-T P.1202 can be applied to the monitoring of performance and quality of experience (QoE) of video services such as IPTV, and has two modes: Mode 1, where the video bitstream are parsed and not decoded into pixels, and mode 2, where the video bitstream are fully decoded into pixels for analyzing.

Table of Contents

1 Scope

Recommendation ITU-T P.1202.2 describes parametric non-intrusive bitstream assessment of video media streaming quality - higher resolution application area (HR), for monitoring the video quality of IP-based video services based on information extracted from the video bitstream encoded with the H.264/AVC baseline video codec. The model are intended especially for the higher resolution application area, including services as IPTV. Recommendation ITU-T P.1202.2 consists of two models corresponding to two modes: mode 1 and mode 2, which both are no-reference (i.e. non-intrusive) models. Mode 1 refers to as parsing mode, the model operates by analysing information in the video bitstream without fully decoding the bitstream (i.e. no pixel information is used) for MOS estimation, and Mode 2 refers to as full decoding mode, in addition to the bitstream information which mode 1 uses, the model can also decode parts or all of the video bitstream (i.e. pixel information is used) for MOS estimation. Further client specific information, such as concealment type, is provided to the algorithm out-of-band, for example in the form of stream specific side information. As output, the model algorithm provides an estimate of the video quality in terms of the 5-point absolute category rating (ACR) mean opinion score (MOS) scale defined in ITU-T P.910.

2 References and abbreviations

2.1 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T P.1202] Recommendation ITU-T P.1202 (2012), Parametric non-intrusive bitstream assessment of video media streaming quality

[ITU-T P.910] Recommendation ITU-T P.910 (2008), Subjective video quality assessment methods for multimedia applications

[ITU-T H.264] Recommendation ITU-T H.264 (2012) Advanced video coding for generic audiovisual services

[ITU-T Rec. H.222.0, ISO/IEC 13818-1] Rec. ITU-T H.222.0 | ISO/IEC 13818-1, Information technology -- Generic coding of moving pictures and associated audio information: Systems

[IETF RFC 3550] IETF RFC 3550 (2003), RTP: A Transport Protocol for Real-Time Applications

[IETF RFC 4566] IETF RFC 4566 (2006), SDP: Session Description Protocol

[IETF RFC 2250] IETF RFC 2250 (1998), RTP Payload Format for MPEG1/MPEG2 Video

2.2 Abbreviations

ACR	Absolute Category Rating
EC MB	Error Concealed Macro Block

GOP	Group Of Pictures
H.264/AVC	H.264 Advanced Video Coding
MB	Macro Block
MOS	Mean Opinion Score
MV	Motion Vector
NAL	Network Abstraction Layer
PLC	Packet Loss Concealment
SPS	Sequence Parameter Set
PPS	Picture Parameter Set
POC	Picture Order Count

3 Model mode 1 description

This section describes the ITU-T Recommendation P.1202.2 model mode 1 and how it is implemented. The block diagram for the model is depicted in figure 3-1. The model takes an H.264/AVC encoded video bitstream and side information (error concealment type, etc.) as input, extracts parameters, also referred to as features, and aggregates them into model parameters which are used to calculate an estimated video quality MOS for the sequence.

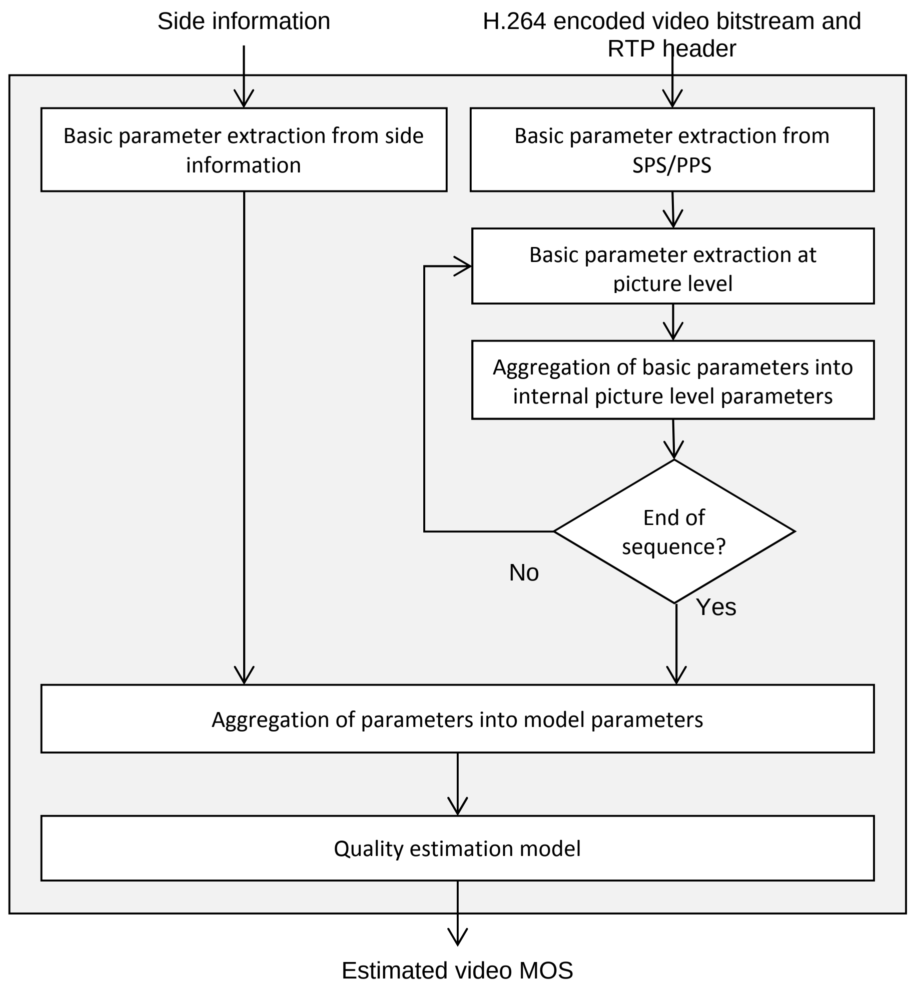

Side information
H.264 encoded video bitstream and RTP header
Basic parameter extraction from side information
Basic parameter extraction from SPS/PPS
Basic parameter extraction at picture level
Aggregation of basic parameters into internal picture level parameters
End of sequence?
No
Yes
Aggregation of parameters into model parameters
Quality estimation model
Estimated video MOS

Figure 3-1 – Block diagram for P.1202.2

The model description is organised according to the flow of parameters in the model and to what degradation module the parameters belong to. Subsection 3.1 describes the extraction of basic parameters from the video bitstream and side information. In the second subsection, 3.2, the algorithms for aggregating the basic parameters into internal parameters at picture level are described. The words picture and frame are used interchangeably throughout this Recommendation. The aggregation of parameters into model parameters is described in subsection 3.3, and finally the quality estimation model is described in subsection 3.4.

Three different types of degradations are detected by the Recommendation ITU-T P.1202.2 model; compression artifacts, slicing artifacts and freezing artifacts. Compression artifacts are introduced due to lossy compression of the encoding process. Slicing artifacts are introduced when packet losses are concealed using a packet loss concealment (PLC) scheme trying to repair erroneous frames. Freezing artifacts are introduced when the PLC scheme of the receiver replaces the erroneous frames (either due to packet loss or error propagation) with the previous *error free* frame until a decoded picture without errors has been received. Since the erroneous frames are not displayed, this type of artifact is also referred to as freezing with skipping. The word *error free* means *no packet losses occurred* throughout the Recommendation.

For each subsection a set of input and output parameters that are used by the algorithms are defined and described. Model coefficients are also given if it is being used by the algorithms.

The flow chart in figure 3-2 describes the flow of parameters in the model and in what subsection each part can be found. Please note that the dotted line is used between section 3.4.1 and 3.4.5, and it means that when artifact is only caused by video compression, the output of section 3.4.1 (*d_compression_quality_value*) is output directly as the overall video quality score. Otherwise, when there is one or more other artifact type, for example, artifact caused by packet losses, the framework module is used to consider degradations caused by various artifact types to generate a total video quality score, i.e. the output of section 3.4.1 (*d_compression_quality_value*), 3.4.2 (*d_slicing_artifact_value*) and 3.4.3 (*d_freezing_artifact_value*) will go into section 3.4.5.

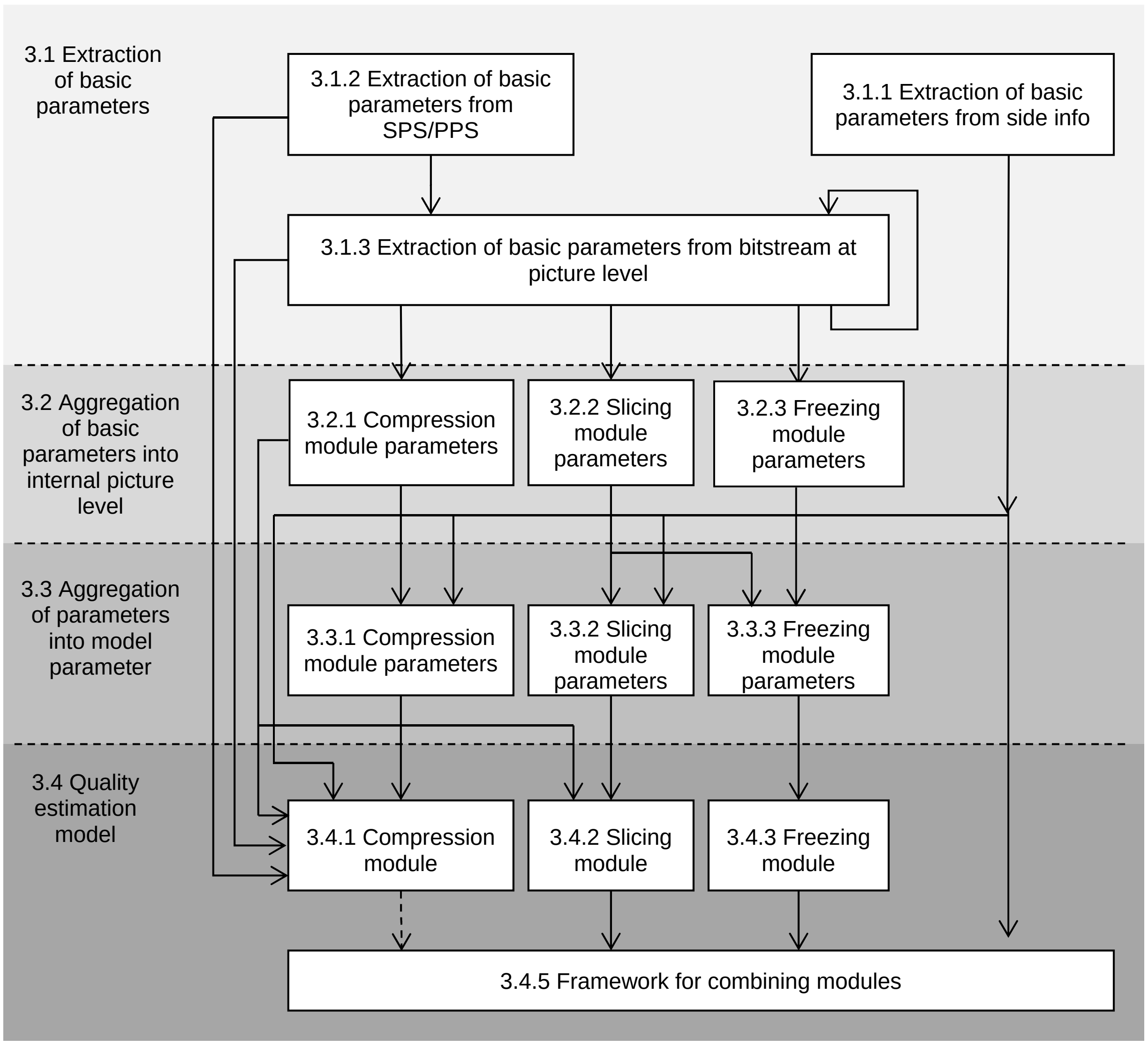

3.1 Extraction of basic parameters
3.1.2 Extraction of basic parameters from SPS/PPS
3.1.1 Extraction of basic parameters from side info
3.1.3 Extraction of basic parameters from bitstream at picture level
3.2 Aggregation of basic parameters into internal picture level
3.2.1 Compression module parameters
3.2.2 Slicing module parameters
3.2.3 Freezing module parameters
3.3 Aggregation of parameters into model parameter
3.3.1 Compression module parameters
3.3.2 Slicing module parameters
3.3.3 Freezing module parameters
3.4 Quality estimation model
3.4.1 Compression module
3.4.2 Slicing module
3.4.3 Freezing module
3.4.5 Framework for combining modules

Figure 3-2 - Flow of parameters in the model

The algorithmic descriptions of the model are described using pseudocode. The variables in the algorithm descriptions are prefixed according to what variable type they pertain to, e.g. *i_nbr_mbs*. The prefixes have the following meaning:

b	–	boolean
c	–	unsigned char

i	–	integer
d	–	double, i.e. floating point value in double precision
f	–	float, i.e. floating point value
s	–	string
p	–	struct or object holding data

The following static code words are used for the macro block data in the pseudocode of the algorithmic descriptions in this section:

INTRA_MB	The macro block is an intra macro block
INTRA16x16_MB	The macro block is an intra macro block with 16x16 intra prediction mode. It is the same as Intra_16x16 in the H.264 Recommendation with any setting of Intra16x16PredMode, CodedBlockPatternChroma and CodedBlockPatternLuma.
MB_ 8X8_TRANSFRM	The prediction residual signal of the macro block is encoded with 8x8 Transform as described in H.264 Recommendation.
UNKNOWN	The type of the macro block is unknown

3.1 Extraction of basic parameters

The extraction of basic parameters is done from side information; from SPS (Sequence Parameter Set) NAL (Network Abstraction Layer) and PPS (Picture Parameter Set) NAL units in the video bitstream, or from the picture level in the video bitstream.

3.1.1 Extraction of basic parameters from side information

The basic parameter extraction from side information has the following input and output parameters:

```
Input  = [side_information]

Output = [f_fps,
          s_video_PLC_mode
          i_numofslices]
```

3.1.1.1 Input parameter description

side_information	Side information containing necessary information for calculating the model score correctly.

3.1.1.2 Output parameter description

f_fps	The number of frames per second for the sequence. *f_fps* should be taken from *side_information* if available, or *f_fps* should be taken from transport packet header.
s_video_PLC_mode	Packet loss concealment method employed in a video decoder. The value is “SLICING” or “FREEZING” or “N/A”.
i_numofslices	The number of slice(s) in a frame. This can be taken from *side_information* if available, otherwise *i_numofslices* can be derived from H.264 bitstream.

3.1.2 Extraction of basic parameters from SPS/PPS

```
Input  = [bitstream]

Output = [i_nbr_mb_ver,
          i_nbr_mb_hor,
          i_nbr_mbs,
          i_pic_init_qp_minus26]
```

3.1.2.1 Input parameter description

bitstream	A video bitstream encoded using the H.264/AVC.

3.1.2.2 Output parameter description

i_nbr_mb_ver	The number of macro blocks in vertical direction for a picture.
i_nbr_mb_hor	The number of macro blocks in horizontal direction for a picture.
i_nbr_mbs	The number of macro blocks in a picture calculated as $i\_nbr\_mbs = i\_nbr\_mb\_hor \cdot i\_nbr\_mb\_ver$
i_pic_init_qp_minus26	Same definition of pic_init_qp_minus26 as defined in H.264/AVC PPS. It can be extract from H.264/AVC bitstream.

3.1.3 Extraction of basic parameters from bitstream at picture level

The extraction of basic parameters from the bitstream at picture level takes part both on RTP level and at picture level in the elementary stream. The extraction of basic parameters from bitstream at picture level has the following input and output parameters:

```
Input  = [i_nbr_mbs,
          RTP_bitstream,
          p_mb_data_array[],
          i_pic_init_qp_minus26]

struct MB_data
{
   i_dct_coef_ac[][],
   i_dct_coef_dc[],
   i_cbp,
   i_mb_type,
   i_qp,
   i_mvx[][],
   i_mvy[][],
   c_predDirection
}

struct MB_data p_mb_data_array[i_nbr_mbs]


Output = [i_frame_num,
          i_displayorder,
          i_received_packets,
          i_lostpackets,
          i_received_bytes,
          i_frame_type,
          b_reflost0,
          b_reflost1,
          i_lostframegap,
          i_slice_qp[],        i_nbr_slice_qp,
          i_slice_size[],
          i_slice_pixel[]
]
```

3.1.3.1 Input parameter description

i_nbr_mbs	See section 3.1.2.2.
RTP_bitstream	The video bitstream encapsulated in RTP packets.
p_mb_data_array[]	Array of structures of type "struct MB_data" containing macro block data. The size of the array is *i_nbr_mbs*. The macro block data in the structs contain the entries described below.
i_pic_init_qp_minus26	See section 3.1.2.2.

The following parameters of *p_mb_data_array[]* are obtained for each MB in a picture.

i_dct_coef_ac[][]	AC components of DCT coefficients of prediction residual. Its dimension is 16x16. First dimension corresponds to 16 4x4 sized-blocks; second dimension corresponds to 16 DCT coefficients of 4x4 DCT transform.
i_dct_coef_dc[]	DC components of DCT coefficients of prediction residual. Its dimension is 16, corresponding to Hadamard Transform coefficients of DC components of DCT transform of each 4x4 size-block.
i_cbp	It has the same meaning as the cbp defined in H.264 syntax.
i_mb_type	Macro block type
i_qp	QP for the macro block
i_mvx[][]	Array of horizontal motion vectors at sub macro block level. The size of the array is 3x16. The first dimension indicates two inter-prediction directions, 0 for forward prediction, 1 for backward prediction, i_mvx[2][] is for normalized motion vectors that will be described in section 3.2.2.3.1. The second dimension corresponds to 16 4x4 blocks. The precision of the motion vector here is the same as that from H.264 syntax.
i_mvy[][]	Array of vertical motion vectors at sub macro block level. The size of the array is 3x16. The first dimension indicates two inter-prediction directions, 0 for forward prediction, 1 for backward prediction. i_mvy[2][] is for normalized motion vectors that will be described in section 3.2.2.3.1. The second dimension corresponds to 16 4x4 blocks.
c_predDirection	prediction direction. 0x00 = intra prediction; 0x01 = forward prediction only; 0x02 = backward prediction only; 0x03 = bi-direction prediction.

3.1.3.2 Output parameter description

The following output parameters are obtained for each parsed picture. Parsed pictures include completely received pictures and partially received pictures.

i_displayorder	Display order of a picture in the bitstream corresponding to the picture number. It is obtained from the POC and/or frame_num syntax for the H.264/AVC bitstream encoded with main or high profile.
i_received_packets	The number of received RTP packets belonging to a picture. A new picture starts by parsing the POC and/or frame_num syntax of the H.264/AVC bitstream. See section 3.1.3.3.1 for calculation.
i_lostpackets	The number of lost RTP packets belonging to a picture. See section 3.1.3.3.1 for calculation.
i_received_bytes	The number of received bytes belonging to a picture in the elementary stream
i_frame_type	The frame type takes one of the following values; 0 = I_frame, 1 = P_frame, 2 = B_frame or 5 = unknown.
b_reflost0	Boolean indicating whether a reference picture in current pictures's reference_picture_list0 is totally lost. Default/initial value is FALSE
b_reflost1	Boolean indicating whether a reference picture in current pictures's reference_picture_list1 is totally lost. Default/initial value is FALSE
i_lostframegap	The number of consecutively totally lost frames from the latest fully/partly received frame. This can be derived from the POC and/or frame_num syntax of H.264.
i_slice_qp[]	Array of correctly decoded slice quantization parameter of a frame, each *i_slice_qp* is in the range of [0, 51] inclusive. The array size is *i_nbr_slice_qp*.
i_nbr_slice_qp	The number of correctly decoded slice quantization parameter of a frame. It increases by 1 if an *i_slice_qp* is obtained within a frame.**i_slice_size[]** Array of number of bytes of a slice of an *error free* Intra frame. Note that, the parameter is only calculated for each *error free* Intra frame.
i_slice_pixel[]	Array of number of pixels of a slice of an *error free* Intra frame. Note that, the parameter is only calculated for each *error free* Intra frame.

3.1.3.3 Algorithms

3.1.3.3.1 Keeping track of pictures and lost packets

A new picture *i* is recorded by detecting the change of POC and/or frame_num syntax from the H.264/AVC bitstream. The display order number value *i_displayorder* calculated from POC and/or frame_num is recorded for the picture *i.* All the RTP packets between the first packet (i.e. the packet having POC and/or frame_num syntax) of the picture *i* and the first packet (i.e. the packet having different POC and/or frame_num syntax) of the next received picture *i+1* are counted as the

RTP packets of the picture i. The number of received bytes, $i\_received\_bytes$, is the sum of elementary stream size in the received packets of the picture.

If one or more consecutive pictures have been lost, it can be detected by detecting the discontinuity in *i_displayorder* of parsed pictures.

The number of lost RTP packets of the picture i is counted based on the discontinuity of sequence numbers of RTP headers. The lost RTP packets between the first received RTP packet of the picture i and the first RTP packet of the next received picture *i+1* are counted into the lost packets of the picture *i*.

3.1.3.3.2 Obtain slice quantization parameter

The parameter, i_slice_qp, is the correctly decoded slice quantization parameter and is obtained per slice within a frame. Correctly decoded means that there is no packet loss occurred between the first transport packet of the slice and the transport packet which contains the bits for acquisition of $slice\_qp\_delta$. $slice\_qp\_delta$ is parsed from the slice header as defined in H.264/AVC.

$$i\_slice\_qp = 26 + i\_pic\_init\_qp\_minus26 + slice\_qp\_delta$$

3.2 Aggregation of basic parameters into internal picture level parameters

This subsection describes how internal picture level parameters are aggregated from the basic parameters extracted from the video bitstream.

3.2.1 Compression module parameters

```
Input  = [i_slice_qp[],
          i_nbr_slice_qp,
          i_slice_size[],
          i_slice_pixel[] ]

Output = [f_frame_content_complexity[],
          i_nbr_error_free_intra_frame,
          i_total_slice_qp,
          i_nbr_total_slice_qp]
```

3.2.1.1 Input parameter description

i_slice_qp[]	See subsection 3.1.3.2
i_nbr_slice_qp	See subsection 3.1.3.2
i_slice_size[]	See subsection 3.1.3.2
i_slice_pixel[]	See subsection 3.1.3.2

3.2.1.2 Output parameter description

f_frame_content_complexity[]	Array of frame content complexity of an *error free* Intra frame. Its array size is *i_nbr_error_free_intra_frame*.
i_nbr_error_free_intra_frame	The number of *error free* Intra frame of the video sequence. It increases by 1 if an *error free* frame is received.
i_total_slice_qp	The total value of *i_slice_qp* of the video sequence. It is the sum of all *i_slice_qp* of the video sequence.
i_nbr_total_slice_qp	The number of *i_slice_qp* of the video sequence. It can be calculated either by summing up *i_nbr_slice_qp* of each frame or increasing by 1 if an *i_slice_qp* is obtained.

3.2.1.3 Algorithms

3.2.1.3.1 Obtain frame content complexity

For each *error free* Intra frame, its frame content complexity (i.e., *f_frame_content_complexity*) is calculated by averaging the value of slice content complexity of all slice(s) (i.e., *f_slice_content_complexity[]*) within the *error free* Intra frame. The *f_slice_content_complexity[]* means an array of slice content complexity of the *error free* Intra frame, and its array size is the number of slice(s) of the *error free* Intra frame, i.e., *f_slice_content_complexity* is calculated per slice within the *error free* Intra frame. For each slice content complexity (i.e., *f_slice_content_complexity*) of the *error free* Intra frame, it is calculated according to its bytes per pixel (i.e., *f_slice_byte_per_pixel*).

The bytes per pixel of a slice (i.e., *f_slice_byte_per_pixel*) is calculated according to its size in bytes (i.e., *i_slice_size*) and pixels (i.e., *i_slice_pixel*) as below:

$$f\_slice\_byte\_per\_pixel_k = \frac{i\_slice\_size_k}{i\_slice\_pixel_k}$$

where k is from 1 to the number of slice(s) of the *error free* Intra frame, inclusive; $i\_slice\_size_k$ means the number of bytes of the k-th slice; $i\_slice\_pixel_k$ means the number of pixels of k-th slice; $f\_slice\_byte\_per\_pixel_k$ means the bytes per pixel of k-th slice.

For implementation, pseudo code is provided below:

```
for( i = 0; i < number_of_slices_of _error_free_intra_frame; i++ )
{
        f_slice_byte_per_pixel[i] = i_slice_size[i] / i_slice_pixel[i];
}
```

The slice content complexity of a slice (i.e. *f_slice_content_complexity*) is calculated according to its correctly decoded quantization parameter of the slice (i.e., *i_slice_qp*) and bytes per pixel of the slice (i.e., *f_slice_byte_per_pixel*) as below:

$$f\_slice\_content\_complexity_k = \mathrm{a}[i\_slice\_qp_k] * f\_slice\_byte\_per\_pixel_k + \mathrm{b}[i\_slice\_qp_k]$$

where k is from 1 to the number of slice(s) of the *error free* Intra frame, inclusive; $i\_slice\_qp_k$ means the correctly decoded slice quantization parameter of the k-th slice; $f\_slice\_byte\_per\_pixel_k$ means bytes per pixel of k-th slice; $f\_slice\_content\_complexity_k$ means the slice content complexity of k-th slice; *a[]* and *b[]* are arrays holding coefficients value for each i_slice_qp respectively and as shown below:

For SD resolution

a[52] = { 24.78954, 24.78954, 25.23854, 25.51193, 25.74990, 25.97533, 26.19479, 26.28303, 26.49158, 26.56645, 26.53197, 26.62563, 26.69239, 26.65409, 26.79309, 26.80578, 26.84816, 27.08741, 27.25370, 27.36097, 27.56078, 27.70162, 27.85621, 28.04059, 28.17621, 28.23445, 28.41471, 28.45078, 28.54265, 28.60014, 28.62930, 28.64529, 28.74102, 28.75523, 28.76358, 28.74681, 28.77488, 28.73642, 28.79531, 28.69430, 28.72766, 28.60666, 28.49484, 28.35642, 28.07614, 27.90134, 27.57123, 27.01405, 26.65987, 26.31439, 25.52575, 25.01169};

b[52] = { 13.39250, 13.39250, 13.97091, 14.53803, 15.25528, 16.13630, 16.99497, 17.66163, 18.80068, 19.89785, 21.20091, 22.86877, 24.44105, 25.98037, 28.04957, 30.07985, 32.07935, 34.30203, 36.32256, 38.18652, 40.93258, 43.77054, 46.53546, 50.53632, 54.36178, 57.82423, 63.29899, 69.18878, 75.07466, 83.80263, 91.47496, 99.18949, 111.47580, 124.34650, 136.49900, 156.17670, 176.23080, 192.16970, 223.83720, 251.77270, 285.92790, 333.53770, 388.41820, 435.09860, 531.05070, 633.24080, 760.16820, 948.15240, 1168.53720, 1361.84570, 1759.43160, 2040.35460};

For 1280x720 resolution

a[52] = { 16.17209, 17.45819, 17.80732, 18.02041, 18.18083, 18.52479, 19.03342, 19.06581, 19.41564, 19.85189, 20.07956, 20.81183, 21.43127, 21.83287, 22.61658, 23.14807, 23.92571, 25.20184, 26.03683, 26.68701, 27.49974, 28.12203, 28.66205, 29.27020, 29.69070, 29.92960, 30.40275, 30.60385, 30.85636, 31.06785, 31.26051, 31.35589, 31.63646, 31.76881, 31.92259, 32.08798, 32.28134, 32.36179, 32.60119, 32.61653, 32.75291, 32.73418, 32.72940, 32.70158, 32.59009, 32.41000, 32.21505, 31.76353, 31.23468, 30.87401, 30.01071, 29.31316 };

b[52] = { 33.81798, 33.05324, 35.11725, 36.95499, 39.10951, 41.62373, 43.87256, 45.95354, 49.32386, 51.87803, 54.92251, 58.42482, 61.62755, 64.56505, 69.19412, 73.35919, 76.10406, 78.96517, 81.95586, 84.59924, 89.05335, 93.59975, 98.31476, 105.41810, 112.34964, 118.73374, 129.00992, 140.01562, 151.12381, 167.62430, 182.02425, 196.08347, 218.72591, 241.16108, 263.35157, 295.99927, 329.06899, 355.66280, 407.64235, 452.09915, 508.72302, 585.36672, 671.43978, 741.49561, 891.18944, 1051.86892, 1246.04333, 1527.50615, 1894.63282, 2204.87735, 2879.95903, 3390.89788 };

For 1920x1080 resolution (1080)

a[52] = { 15.75673, 16.17239, 17.33657, 18.09218, 18.78856, 19.85244, 20.94081, 21.42377, 25.25608, 25.36929, 25.37671, 25.59413, 25.77414, 25.89431, 26.16539, 26.37098, 26.71202, 27.45373, 27.99336, 28.43923, 29.01115, 29.49924, 29.89337, 30.32379, 30.59313, 30.74944, 31.01314, 31.10389, 31.21737, 31.28295, 31.38585, 31.36863, 31.44693, 31.40169, 31.43938, 31.39075, 31.36072, 31.33672, 31.26816, 31.16160, 31.03165, 30.80631, 30.57609, 30.36353, 30.06076, 29.62381, 29.37353, 29.05716, 28.60942, 28.52338, 28.40104, 28.52280};

b[52] = { 25.92973, 26.42403, 26.72231, 27.10874, 27.55908, 27.59167, 27.40409, 27.63129, 21.08740, 22.32786, 23.78112, 25.55635, 27.25511, 28.80079, 31.33600, 33.71534, 35.51380, 37.14249, 38.57997, 39.75292, 41.50986, 43.25411, 45.08496, 47.92251, 50.97660, 53.82247, 58.50549, 64.00109, 69.59487, 78.31654, 84.35147, 92.89916, 105.12040, 119.83478, 131.13182, 152.46046, 175.28796, 191.40711, 231.17849, 262.14953, 311.33306, 374.98524, 454.98602, 524.68907, 656.91124, 830.55605, 990.09180, 1196.94617, 1493.32352, 1667.34794, 1966.34090, 2099.62991};

For implementation, pseudo code is provided below:

```
for( i = 0; i < number_of_slices_of _error_free_intra_frame; i++ )
{
        f_slice_content_complexity[i] = a[i_slice_qp[i]] * f_slice_byte_per_pixel[i] + b[i_slice_qp[i]];
}
```

For each *error-free* Intra frame, its frame content complexity (i.e., f_frame_content_complexity) is calculated according to its slice content complexity of all slice(s) (i.e., *f_slice_content_complexity[]*) and its number of slice(s) as below:

$$f\_frame\_content\_complexity = \frac{\sum_{\mathrm{k}=1}^{\mathrm{Num}} f\_slice\_content\_complexity_{\mathrm{k}}}{Num}$$

where Num is the number of slice(s) of the *error free* Intra frame; k is from 1 to the number of slice(s) of the *error free* Intra frame, inclusive; $f\_slice\_content\_complexity_{\mathrm{k}}$ means the slice content complexity of the k-th slice.

For implementation, pseudo code is provided below:

```
float f_total_slice_content_complxity = 0.0f;
for( i = 0; i < number_of_slices_of _error_free_intra_frame; i++ )
{
        f_total_slice_content_complexity += f_slice_content_complexity[i]
}
f_frame_content_complexity = f_total_slice_content_complexity / num_of_slice_of_error_free_intra_frame
```

3.2.2 Slicing module parameters

```
Input  = [i_nbr_mbs,
          f_fps,
          i_displayorder,
          i_received_packets,
          i_lostpackets,
          i_received_bytes,
          i_frame_type,
          b_reflost0,
          b_reflost1,
          i_lostframegap,
          p_mb_data_array[]]


Output = [i_processed_frames,
          p_frame_parameters[]]

struct Frame_parameters
{
       i_nbr_ecmbs,
       i_frame_type,
       i_ecdist,
       b_reflost0,
       b_reflost1,
       i_refFramesIdx[]
       d_IntraRatio,
       i_totalpackets,
       i_lostpackets,
       i_totalbytes,
       i_avgbytesPerframe,
       d_avgmv,
       d_residual_var,
       b_fade_pic,
       b_scut_candidate,
       b_scut_loss,
       d_LoVA,
       struct MB_parameters *p_mb_parameters
}

struct Frame_parameters * p_frame_parameters[MAXFRAMES]

struct MB_parameters
{
       d_mvmedian,
       d_mvmedian_ref0,
       d_mvmedian_ref1,
       b_ec,
       b_mvexist,
       c_predDirection,
       b_intramode,
       d_residual_var,
       d_lova
}
```

3.2.2.1 Input parameter description

i_nbr_mbs	See subsection 3.1.2.2
f_fps	See subsection 3.1.1.2
p_mb_data_array[]	See subsection 3.1.3.1

For all other input parameters see subsection 3.1.3.2.

3.2.2.2 Output parameter description

i_processed_frames	The number of parsed frames. This variable is increased by one when a frame is parsed by the H.264/AVC parser by recording the change of i_displayorder of adjacent frames. If there is discontinuity in

	i_displayorder, it means that the packets of one or several frame(s) are lost totally, thus increase i_processed_frames correspondingly. The details are shown in pseudo code below.
p_frame_parameters[]	Array holding parameters for each parsed picture.

The following output parameters are calculated or saved from parser information for each parsed picture in *p_frame_parameters[]*:

i_nbr_ecmbs	The number of error concealed MBs in a frame.
i_frame_type	Frame type for the picture. Default/initial value is 5, i.e. unknown frame type.
i_ecdist	An error concealment distance, defined as a distance, in display order, between the to-be-concealed picture and the concealing picture (i.e., the picture used for error concealment). It is explained in subsection 3.2.2.3 how this parameter is derived. Default/initial value is a larger value, e.g., 0xFFFF.
b_reflost0	See subsection 3.1.3.2. Default/initial value is FALSE.
b_reflost1	See subsection 3.1.3.2. Default/initial value is FALSE.
i_refFramesIdx[]	The display order of the immediate forward [0] and/or backward [1] reference frame of current frame. Its dimension is 2.
d_IntraRatio	The ratio of the number of intra macro blocks to all received macro blocks, i.e*., i_nbr_intraMB / (i_nbr_mbs – i_nbr_ecmbs).*
i_totalpackets	The total number of packets belonging to a frame. It includes the received packets and the estimate of the lost packets of the frame.
i_lostpackets	See subsection 3.1.3.2
i_totalbytes	The number of total bytes belonging to a frame. It includes the received bytes and the estimate of the bytes of lost packets.
i_avgbytesPerframe	The average number of bytes per frame calculated in a sliding window of L=10 frames. Initial value is 0.
d_avgmv	Average motion of the inter-prediction MBs in the frame. Initial value is 0.
d_residual_var	Average variance of the prediction residual signal in the frame. Initial value is 0.
b_fade_pic	Boolean indicating whether a frame have video content of gradual transition effects. The default/initial value is FALSE.
b_scut_candidate	Boolean indicating whether this frame is a scene cut candidate frame. The default/initial value is FALSE.
b_scut_loss	Boolean indicating whether a frame having packet loss is a scene cut frame. The scene cut frame can be current frame or the immediately previous reference frame that is lost completely. The initial value is FALSE.
d_LoVA	The level of visible artifacts in a frame.
p_mb_parameters[]	Pointer to an array of MB parameters. Its size is *i_nbr_mbs*.

The following output parameters are calculated or saved for each MB in *p_mb_parameters[]*.

d_mvmedian	The magnitude of the median of the available motion vectors of the macro block and its immediate adjacent macro blocks.
d_mvmedian_ref0	The magnitude of the median of the macro block's motion vectors pointed to the lost immediately forward reference frame. The default value is zero.
d_mvmedian_ref1	The magnitude of the median of the macro block's motion vectors pointed to the lost immediately backward reference frame. The default value is zero.
b_ec	Boolean indicating whether error concealment is applied to the macro block or not. Initial value is FALSE.
b_mvexist	If the *i_mb_type* is intra mode or the MB is error concealed, its value is FALSE. Otherwise, TRUE. Initial value is FALSE.
c_predDirection	See subsection 3.1.3.2. Default/initial value is 0x00.
b_intramode	Boolean indicating if this MB is encoded with intra mode.
d_residual_var	Variance (i.e. AC energy) of the prediction residual signal of the MB. Initial value is 0.
d_lova	The level of visible artifacts of the MB

3.2.2.3 Algorithms

The main components are shown in figure 3-3. First, a parsing of the sequence is made to derive frame-level parameters in structure *p_frame_parameters[]* and MB-level parameters in structure *p_mb_parameters[]*. Secondly, detect if a frame is a gradual transition frame and set *b_fade_pic* correspondingly. For each of the frames whose *i_lostpackets > 0*, detect if it is candidate scene cut frame and set *b_scut_candidate* correspondingly. If *b_scut_candidate* is true and the frame is not a gradual transition frame, further detect if the frame is a frame having scene cut artifact and set *b_scut_loss* correspondingly.

The important output parameters in this stage consist of MB-level parameters i.e. *d_mvmedian*, *d_residual_var*, and frame-level parameters i.e. *b_fade_pic, b_scut_candidate, b_scut_loss.* These parameters will be used to estimate the visible artifact level due to packet loss in each frame of the sequence. Then the frames' visible artifact levels are aggregated into a sequence-level visible artifact metric.

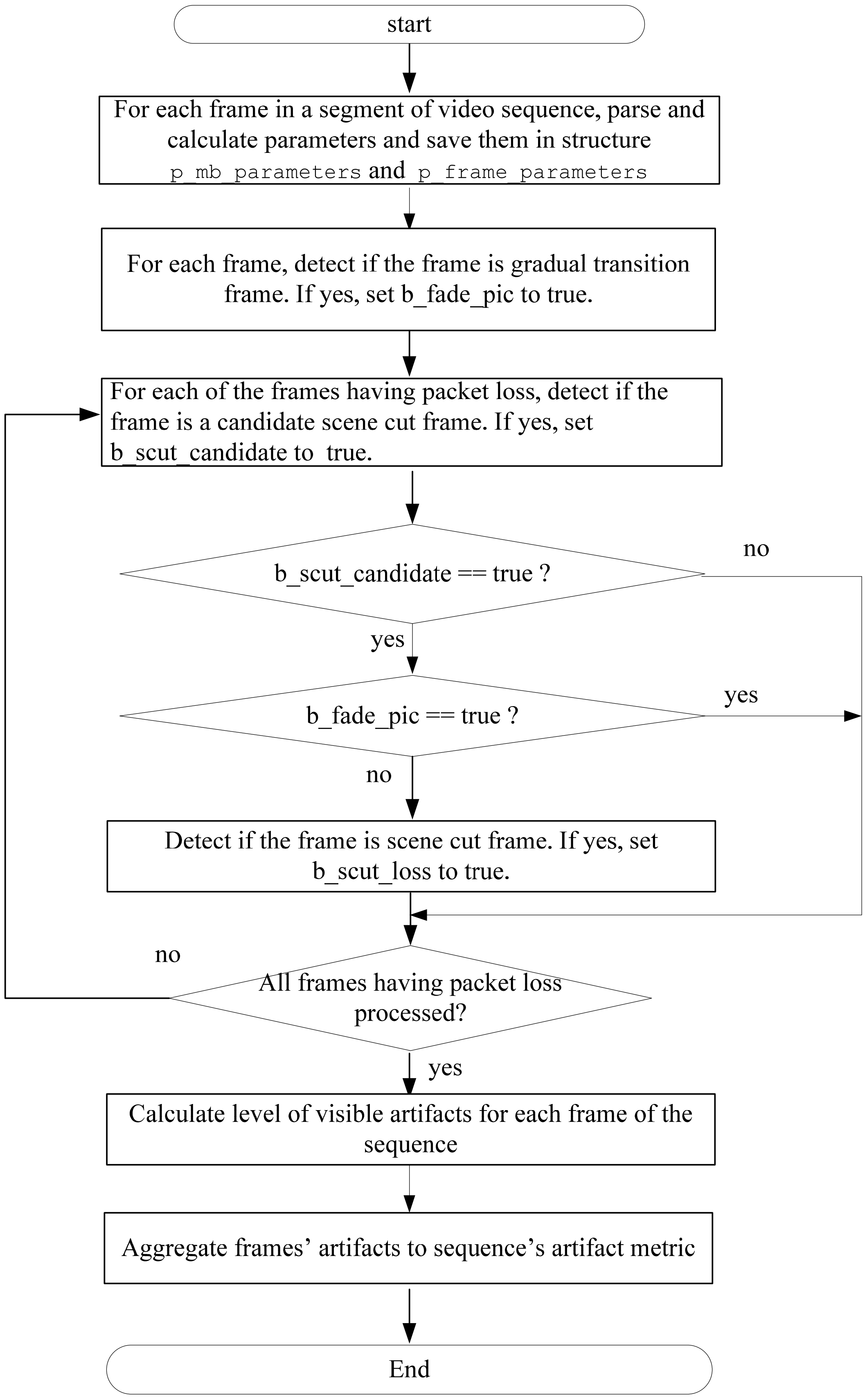

start
For each frame in a segment of video sequence, parse and calculate parameters and save them in structure p_mb_parameters and p_frame_parameters
For each frame, detect if the frame is gradual transition frame. If yes, set b_fade_pic to true.
For each of the frames having packet loss, detect if the frame is a candidate scene cut frame. If yes, set b_scut_candidate to true.
b_scut_candidate == true ?
no
yes
b_fade_pic == true ?
yes
no
Detect if the frame is scene cut frame. If yes, set b_scut_loss to true.
no
All frames having packet loss processed?
yes
Calculate level of visible artifacts for each frame of the sequence
Aggregate frames' artifacts to sequence's artifact metric
End

Figure 3-3 - Overview of components

3.2.2.3.1 Obtain motion and residual energy of MBs

d_mvmedian is the magnitude of the median of the available motion vectors of 4x4 blocks of the MB and its eight or four immediate adjacent MBs. In this implementation, using four immediate adjacent MBs can obtain similar performance as using all eight adjacent MBs. The term "available motion vector" means that the corresponding MB is correctly decoded and is an inter-predicted MB. For a partition that contains multiple 4x4 blocks, the blocks get the MV of the containing partition. For example, the two 4x4 blocks within a 4x8 partition of a MB has the same motion vector as that of the 4x8 partition.

For a MB having no available motion vector in itself or in its four immediate adjacent MBs, its *d_mvmedian* needs to be estimated. The estimation of *d_mvmedian* for a MB depends on the frame type of the frame that the MB belongs to. If the MB is in an Intra frame, *d_mvmedian* of the MB is set to the *d_mvmedian* of a collocated MB in the closest decoded reference frame. If the MB is in a non-Intra frame, there are several cases. If the MB is correctly decoded and is an intra MB, then its motion vector is set to zero. Otherwise, motion vectors of the MBs at slice boundaries are set to that of the spatially closest upper MB, and motion vectors of internal MBs in a slice are set to that of a previous reference frame. The reference frame means I-frame, or P-frame, or reference B-frame.

For a block of an inter-predicted MB, it may have different prediction directions or refer to different prediction lists. For example, a MB in a B-frame may use forward prediction only, backward prediction only, or bi-directional prediction. MBs in one frame may also have different reference frames due to the use of multiple reference frames. Thus, we normalize the motion vectors with the distance between the current frame and the reference frame, and save the normalized results in *p_mb_data_array*[].*i_mvx*[0/1][] and *p_mb_data_array*[].*i_mvy*[0/1][]. To unify the meanings of motion vectors in B frames and P frames, we further normalize the motion vectors with the reference direction and save the results in *p_mb_data_array*[].*i_mvx*[2][] and *p_mb_data_array*[].*i_mvy*[2][]. That is,

$$i\_mvx[2][k] = \begin{cases} i\_mvx[0][k], & \text{forward prediction mode} \\ -1 * i\_mvx[1][k], & \text{backward prediction mode} \\ \dfrac{i\_mvx[0][k] - i\_mvx[1][k]}{2}, & \text{bi}-\text{directional prediction} \end{cases}$$

Apply the same operation for *i_mvy*[2][k], k=0,…,15.

i_ecdist is a distance, in display order, between the to-be-concealed picture and the concealing picture (i.e., the picture used for error concealment). In this implementation, the concealing picture is the immediately previous decoded forward reference picture of the to-be-concealed picture. The calculation is given in pseudo code.

These two parameters, *d_mvmedian* and *i_ecdist,* are used in the next section to estimate the level of visible artifacts in an error concealed (EC) MB. An EC MB may be a lost MB or an MB that cannot be parsed due to loss of sync of H.264 syntax.

The residual energy *d_residual_var* of a non-EC MB is calculated as the variance of the energy of de-quantized transform coefficients. For an EC MB, its *d_residual_var* is set to the corresponding value of the collocated MB in its previous reference frame.

The pseudo code functions for obtaining the MB-level parameters are:

```
void get_MB_parameters (int frmidx)
{
  struct MB_parameters *currfrm_mbparameter, *prevfrm_mbparameter;
  currfrm_mbparameter = p_frame_parameters[frmidx]-> p_mb_parameters;
  prevfrm_mbparameter = p_frame_parameters[i_prevfrm]-> p_mb_parameters;

  /*get d_mvmedian, d_mvmedian_ref0,d_mvmedian_ref1, d_residual_var of a MB*/
  if (i_frame_type == I_frame)
  {
    for (m=0;m< i_nbr_mbs;m++)
    {
      currfrm_mbparameter[m].b_intramode =
          (p_mb_data_array[m].i_mb_type == INTRA_MB);
      currfrm_mbparameter[m].b_ec = (p_mb_data_array[m].i_mb_type ==UNKNOWN);
      if (currfrm_mbparameter[m].b_ec == false)
         get_DCT_energy(p_mb_data_array[m],currfrm_mbparameter[m]);
      else
      {
         // Variance of residual signal is used for scene cut detection, thus
         // fill in with value of collocated MB in previous I-frame
         int k = i_displayorder -1;
         while (p_frame_parameters[k]->frame_type != I_frame)
                  k--;
         currfrm_mbparameter[m].d_residual_var =
                 p_frame_parameters[k]-> p_mb_parameters[m].d_residual_var;
       }
       currfrm_mbparameter[m].d_mvmedian =
                prevfrm_mbparameter[m].d_mvmedian;
    }
  }
  else //inter-predicted frame
  {
    for (m=0;m< i_nbr_mbs;m++)
    {
       currfrm_mbparameter[m].b_intramode =
             (p_mb_data_array[m].i_mb_type == INTRA_MB);
       currfrm_mbparameter[m].b_ec = (p_mb_data_array[m].i_mb_type == UNKNOWN);
       if (currfrm_mbparameter[m].b_ec == false)
       {
         if (p_mb_data_array[m].i_mb_type != INTRA_MB)
         {
          currfrm_mbparameter[m].c_predDirection =
               p_mb_data_array[m]. c_predDirection;
           currfrm_mbparameter[m].b_mvexist = true;
          currfrm_mbparameter[m].d_mvmedian_ref0 =
          sqrt(pow(getMedian(p_mb_data_array[m].i_mvx[0]),2) +
                     pow(getMedian(p_mb_data_array[m].i_mvy[0]),2));
          currfrm_mbparameter[m].d_mvmedian_ref1 =
          sqrt(pow(getMedian(p_mb_data_array[m].i_mvx[1]),2) +
                     pow(getMedian(p_mb_data_array[m].i_mvy[1]),2));
         }
         get_DCT_energy(p_mb_data_array[m], currfrm_mbparameter[m]);
       }
       else
       {
         currfrm_mbparameter[m].d_residual_var =
             prevfrm_mbparameter[m].d_residual_var;
       }
       if (MB m has available motion vectors)
       {
            currfrm_mbparameter[m].d_mvmedian =
                   sqrt(pow(getMedian(p_mb_data_array[m].i_mvx[2]),2) +
                        pow(getMedian(p_mb_data_array[m].i_mvy[2]),2));
        }
        else if (currfrm_mbparameter[m].b_intramode)
        {
            currfrm_mbparameter[m].d_mvmedian = 0;
        }
        else if (i_numofslices > 1 && m - i_nbr_mb_hor >= 0)
        {
            currfrm_mbparameter[m].d_mvmedian =
            prevfrm_mbparameter[m - i_nbr_mb_hor].d_mvmedian;
        }
        else
        {
            currfrm_mbparameter[m].d_mvmedian =
                   prevfrm_mbparameter[m].d_mvmedian;
        }
     }
  }
}
```

`getMedian()` function is defined as:

$$\mathrm{getMedian}(\{x_n\}) = \begin{cases} x_k, & n = 1, \dots, 2k-1, k \geq 1 \\ \frac{1}{2} \times (x_k + x_{k+1}), & n = 1, \dots, 2k, k \geq 1 \end{cases}$$

```
void get_DCT_energy(p_mb_data_array[m],currfrm_mbparameter[m])
{
    double dc = 0;
    double energy_MB = 0;

    if ((i_cbp)%16 > 0 || i_mb_type == INTRA16x16_MB)
    {
        if (MB_8X8_TRANSFRM)
        {
            for (int i=0; i<4; i++)
            {
               for (int j=0; j<64; j++)
                    energy_MB += pow(i_dct_coef_8x8[i][j], 2.0);
               dc += i_dct_coef_8x8[i][0];
            }
            dc /= (4*8);
            currfrm_mbparameter[m]->d_residual_var =
                    (energy_MB/256 – dc*dc)* Qstepsqr[i_qp];

        }
        else
        {
           for (int i=0;i<16;i++)
           {
               for (int j=1; j<16; j++)
                    energy_MB += pow(i_dct_coef_ac[i][j], 2.0);
           }
           if (i_mb_type == INTRA16x16_MB)
           {
               for (int i=0;i<16;i++)
               {
                    energy_MB += pow(i_dct_coef_dc[i], 2.0);
               }
               dc = i_dct_coef_dc[0] /16.0;
            }
            else
            {
                for (int i=0; i<16;i++)
                {
                     energy_MB += pow(i_dct_coef_ac[i][0], 2.0);
                     dc += i_dct_coef_ac[i][0];
                }
                dc /= (16*4);
            }
            currfrm_mbparameter[m]->d_residual_var =
              (energy_MB/256 - dc*dc)*Qstepsqr(i_qp);
        }
    }
}
```

$\mathrm{Qstepsqr(QP)}$ is the square of quantization step, Qstep(QP). QP is the quantization parameter of the MB, which comes from syntax parsing. The quantization step is calculated as

$$\mathrm{Qstep(QP)} = \mathrm{Qstep(QP\%6)} \cdot 2^{\mathrm{floor(QP/6)}}$$

$$\mathrm{Qstep}(0) = \frac{40}{2^6} = 0.625,\;\; \mathrm{Qstep(n)} = \mathrm{Qstep(n-1)} \cdot \sqrt[6]{2},\;\; \mathrm{n} = 1, \ldots, 5$$

The QP takes value from 0 to 51.

3.2.2.3.2 Obtain frame-level parameters

For frame *frmidx* having packet losses or having immediately preceding frame(s) lost, we calculate a parameter, *avgpackets_perframe*, by averaging *i_totalpackets* of the previous (non-I) frames in a sliding window of length N (here N = 10) frames, that is, *avgpackets_perframe* is defined as the average (estimated) number of transmitted packets preceding the current frame.

In addition, the average number of bytes per frame, *i_avgbytesPerframe*, may be calculated by averaging *i_totalbytes* of immediately previous frames in a sliding window of N frames. For correctly received frame, *i_totalbytes* is set to *i_received_bytes*.

The pseudocode functions for calculating the frame-level parameters *i_avgbytesPerframe*, *i_totalbytes* are given below.

```
void get_frame_parameters(int frmidx)
{
  p_frame_parameters[frmidx]->i_frame_type = i_frame_type;
  p_frame_parameters[frmidx]->b_reflost0 = b_reflost0;
  cb_reflost1 = b_reflost1;

  /*calculate feature i_ecdist*/
  if (i_frame_type == B_frame)
       p_frame_parameters[frmidx]->i_ecdist = frmidx -
           displayorder of its immediate forward reference frame;
  else
       p_frame_parameters[frmidx]->i_ecdist = frmidx -
           displayorder of the immediate forward non-B reference frame;

 /*get i_refFramesIdx*/
 if (i_frame_type == I_frame)
     p_frame_parameters[frmidx]->i_refFramesIdx[0] =
        displayorder of its immediate previous reference frame;
  else /* i.e.,inter-predicted frame*/
```

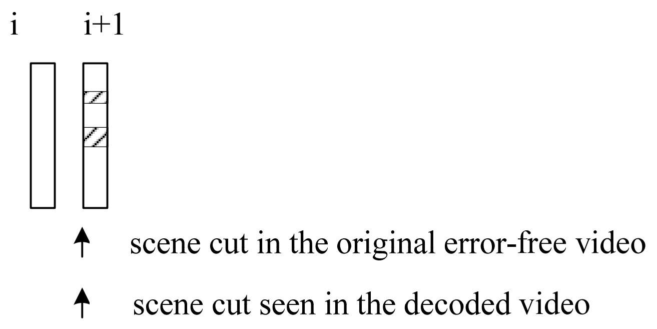
```
   p_frame_parameters[frmidx]->i_refFramesIdx[0] =
      displayorder of its immediate forard reference frame;
if (i_frame_type == B_frame)
   p_frame_parameters[frmidx]->i_refFramesIdx[1] =
     displayorder of its immediate backward reference frame;

p_frame_parameters[frmidx]->i_lostpackets = i_lostpackets;
struct MB_parameters *currfrm_mbparameter;
currfrm_mbparameter = p_frame_parameters[frmidx]-> p_mb_parameters;
acenergy = 0; cnt_goodmbs = 0;
avgmotion = 0; cnt_mv = 0;
intraMBs = 0;
for (int m=0;m<i_nbr_mbs;m++)
{
   if (currfrm_mbparameter[m].b_ec == false)
   {
      cnt_goodmbs ++;
      acenergy += currfrm_mbparameter[m].d_residual_var;
      if (currfrm_mbparameter[m].b_mvexist)
      {
        avgmotion += currfrm_mbparameter[m].d_mvmedian;
        cnt_mv ++;
      }
      intraMBs += (currfrm_mbparameter[m].b_intramode?1:0);
   }
}
p_frame_parameters[frmidx]->i_nbr_ecmbs = i_nbr_mbs - cnt_goodmbs;
p_frame_parameters[frmidx]->d_IntraRatio = intraMBs/cnt_goodmbs;
p_frame_parameters[frmidx]->d_residual_var = acenergy/cnt_goodmbs;
if (cnt_mv > i_nbr_mbs/2)
   p_frame_parameters[frmidx]->d_avgmv = avgmotion/cnt_mv;
else
   p_frame_parameters[frmidx]->d_avgmv =
       p_frame_parameters[i_prevfrm]->d_avgmv;

p_frame_parameters[frmidx]->i_totalpackets = i_received_packets;
p_frame_parameters[frmidx]->i_totalbytes = i_received_bytes;

// if loss occurs, calculate i_avgbytesPerframe and avgpackets_perframe in a
// sliding window of 10 frames;
if (p_frame_parameters[frmidx]->i_lostpackets > 0)
{
  winLen = 10;
  i_dividnum = 0;
  avgbytes_perframe = 0;
  avgpackets_perframe = 0;
  for (int s = frmidx – i_lostframegap - winLen; s < frmidx; s++)
  {
    if(s < 0) break;
    if(p_frame_parameters[s]->i_frame_type == B_frame)
    {
       i_dividnum ++;
       avgbytes_perframe += p_frame_parameters[s]->i_totalbytes;
       avgpackets_perframe += p_frame_parameters[s]->i_totalpackets;
    }
  }
  if (i_dividnum >0)
  {
     avgbytes_perframe /= i_dividnum;
     avgpackets_perframe /= i_dividnum;
     p_frame_parameters[frmidx]->i_avgbytesPerframe = avgbytes_perframe;
    }

    p_frame_parameters[frmidx]->i_totalpackets = i_received_packets  +
         max(0,(i_lostpackets - avgpackets_perframe * i_lostframegap));
    p_frame_parameters[frmidx]->i_totalbytes = i_received_bytes +
         max(0,i_lostpackets * avgbytes_perframe/avgpackets_perframe -
         avgbytes_perframe * i_lostframegap);
  }
}
```

The codes for obtaining the above MB-level and frame-level parameters in a sequence is:

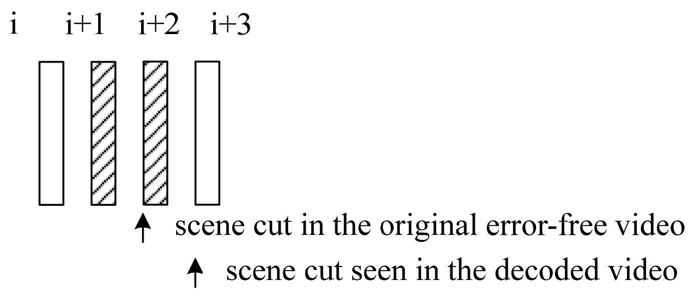
```
void get_internal_parameters ()
{
   i_processed_frames = 0;
   while (get_next_pic_data is successful)
   {
      get_MB_parameters(i_displayorder);
      get_frame_parameters(i_displayorder);
       /* A global variable record the previous parsed frame. It will be used in
      get_mb_parameters() function and get_frame_parameters() function */
      i_processed_frames ++;
   }
   // Count in lost frames. Assumption: the last frame of a sequence isn't lost
   for (i=0;i<i_processed_frames; i++)
   {
     if (p_frame_parameters[i]->i_frame_type == UNKNOWN)
        i_processed_frames ++;
   }
}
```

3.2.2.3.3 Detect scene cut candidate frames where packet loss occurs

When there is a significant scene change between two adjacent pictures i and i+1 and packet loss occurs in the second picture i+1 of the two adjacent pictures as shown in figure 3-4(a), the concealed second picture i+1 will have very strong visible artifacts because temporal error concealment method is used in this implementation. These artifacts are defined as scene cut artifacts in this Recommendation. Scene cut artifacts may also be detected in the first received picture (e.g., picture i+3 in figure 3-4(b)) after one or more subsequent pictures (e.g., pictures i+1, i+2) have been lost completely and the first received picture (e.g., i+3) is compressed using a lost scene cut picture (e.g., picture i+2) as reference picture. This is shown in figure 3-4(b).


i
i+1
scene cut in the original error-free video
scene cut seen in the decoded video

(a)


i
i+1
i+2
i+3
scene cut in the original error-free video
scene cut seen in the decoded video

(b)

Figure 3-4 - Pictorial examples depicting how scene cut artifacts relate to scene cut

Since scene cut artifacts occur at partially received scene cut frames or at frames referring to lost scene cut frames, the frames with or surrounding packet losses may be regarded as potential scene cut artifact locations. Since a scene cut frame is usually encoded as an I-frame, a partially received I-frame may be marked as a candidate location for scene cut artifacts. A scene cut frame may also be encoded as a non-intra (for example, a P-frame). Scene cut artifacts may also occur in such a frame when it is partially received. A frame may also contain scene cut artifacts if it refers to a completely lost scene cut frame. When the total bytes of the frame (*i_totalbytes*) is much larger than the average bytes per frame in a sliding window near the frame (*i_avgbytesPerframe*), the frame may be identified as a candidate scene cut frame in the decoded video.

The flow chart for deriving *b_scut_candidate* is shown in figure 3-5.

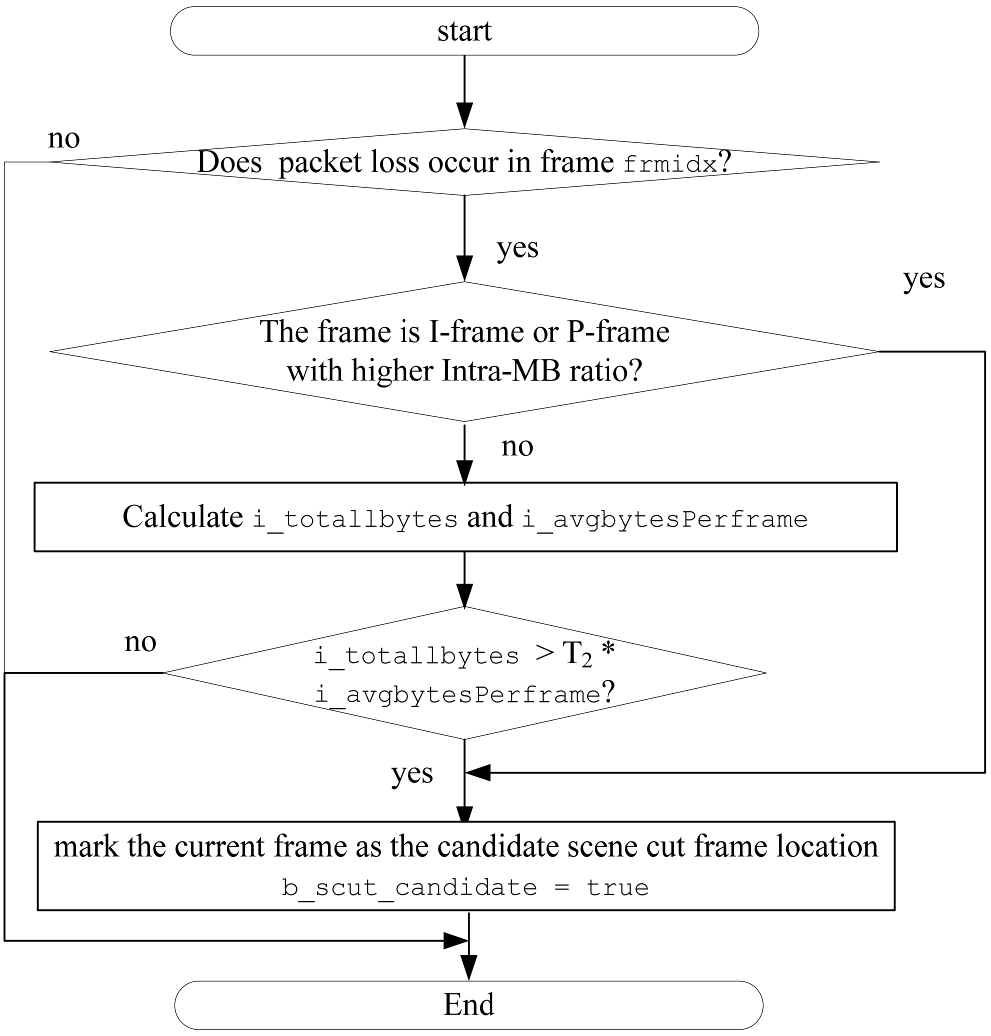

start
Does packet loss occur in frame frmidx?
no
yes
The frame is I-frame or P-frame with higher Intra-MB ratio?
yes
no
Calculate i_totallbytes and i_avgbytesPerframe
i_totallbytes > T2 * i_avgbytesPerframe?
no
yes
mark the current frame as the candidate scene cut frame location
b_scut_candidate = true
End

Figure 3-5 - Flow chart of detecting scene cut candidate frame

The pseudocode to derive *b_scut_candidate* is:

```
void mark_candidate_scut(int frmidx)
{
  // Note: p_frame_parameters[frmidx]-> is omitted for each of the variables
  // below.
  double INTRA_IP = 0.7;
  double T2 = 4.0;

  if (i_lostpackets == 0)
     return;

  if (i_nbr_ecmbs > 0 && i_frame_type == I_FRAME)
  {
     b_scut_candidate = true;
       return;
  }
  else  if(i_nbr_ecmbs  >  0  &&  i_frame_type  ==  P_FRAME  &&(d_IntraRatio  >
INTRA_IP))
  {
     b_scut_candidate = true;
```

```
        return;
    }
    else if (b_reflost0 > 0 || b_reflost1 > 0)
    {
      // i_totallbytes and i_avgbytesPerframe are calculated together with other
      // frame-level parameters in function get_frame_parameters()
      if (i_totalbytes/i_avgbytesPerframe > T2)
      {
        b_scut_candidate = true;
      }
    }
}
```

3.2.2.3.4 Detect scene cut picture with artifacts

The flow chart for deriving *b_scut_loss* is shown in figure 3-6. Scene cut artifact is detected for a candidate frame with *b_scut_candidate* equal to TRUE and *b_fade_pic* equal to FALSE.

When the candidate scene cut frame is an I-frame or P-frame with higher intra MB ratio, the prediction residual energy difference between the picture and a preceding I-frame or P-frame with higher intra MB ratio is calculated. If the difference between the energy factors is T1 times larger than the larger energy factor, the candidate I-frame is detected as a scene cut frame in the decoded video.

The prediction residual energy change or motion change around a scene change is often greater. Thus, when the residual energy difference or motion difference exceeds a threshold, the candidate frame is detected as having scene cut artifacts.

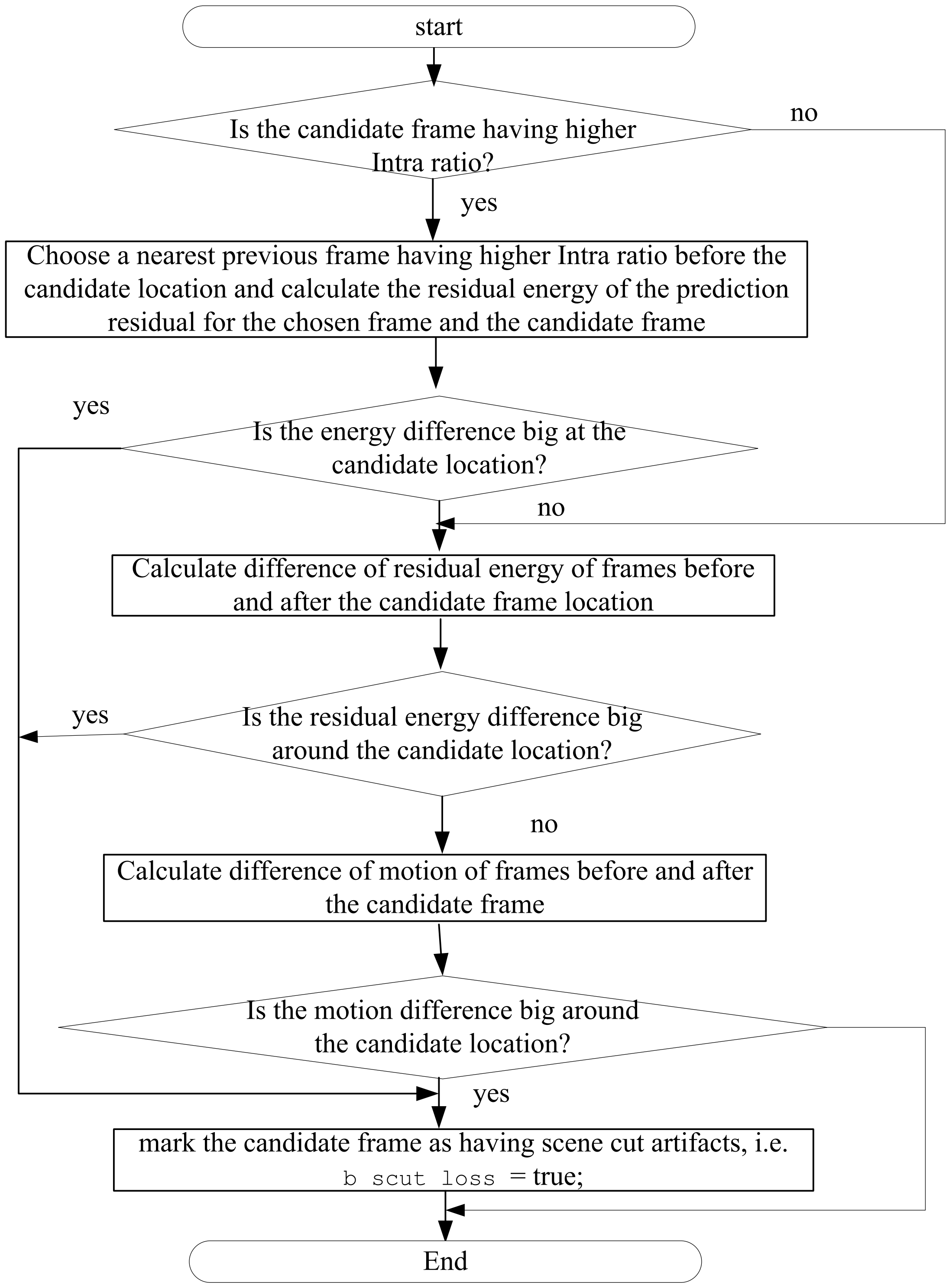

start
Is the candidate frame having higher Intra ratio?
no
yes
Choose a nearest previous frame having higher Intra ratio before the candidate location and calculate the residual energy of the prediction residual for the chosen frame and the candidate frame
yes
Is the energy difference big at the candidate location?
no
Calculate difference of residual energy of frames before and after the candidate frame location
yes
Is the residual energy difference big around the candidate location?
no
Calculate difference of motion of frames before and after the candidate frame
Is the motion difference big around the candidate location?
yes
mark the candidate frame as having scene cut artifacts, i.e. b_scut_loss = true;
End

Figure 3-6 - Flow chart for detection scene cut frame

The codes to derive *b_scut_loss* is:

```
void detect_scenecut (int frmidx, int currfrmid)
{
   double INTRA_IP = 0.7;
   int Lwin = f_fps * 1.0;

  /* It is less likely to have two scene changes in 1 seconds */
  for (i= displayorder - Lwin; i< displayorder + Lwin; i++)
```

```
{
  if (p_frame_parameters[i]->b_fade_pic > 0)
     return;
}
 /*judge by energy difference between current and previous intra frames*/
 if (p_frame_parameters[frmidx]->d_IntraRatio >= INTRA_IP &&
       p_frame_parameters[frmidx]->i_nbr_ecmbs <= i_nbr_mbs/3)
 {
  double thrd_T1      = 0.45;
 double thrd_Ienergy = 36;

  // Find immediately previous received frame with high intra ratio
  int idx_prevIframe = i_displayorder -1;
  while (idx_prevIframe >= 0)
  {
        if ((p_frame_parameters[idx_prevIframe]-> d_IntraRatio  >INTRA_IP) &&
             p_frame_parameters[idx_prevIframe]->i_nbr_ecmbs <= i_nbr_mbs/3)
             break;
        idx_prevIframe --;
  }
  // Calculate residual energy difference
  struct MB_parameters * currfrm_mbparameter, * prevfrm_mbparameter;
  currfrm_mbparameter = p_frame_parameters[frmidx]->p_mb_parameters;
  prevfrm_mbparameter = p_frame_parameters[idx_prevIframe]->p_mb_parameters;
  mbcnt = 0;
  currenergy = 0;
  prevenergy =0;
  for (int m=0; m< i_nbr_mbs; m++)
  {
      if (currfrm_mbparameter[m].b_ec == false)
      {
          mbcnt ++;
          currenergy += currfrm_mbparameter[m].d_residual_var;
          prevenergy += prevfrm_mbparameter[m].d_residual_var;
      }
  }
  currenergy /= mbcnt;
  prevenergy /= mbcnt;
  if (max(currenergy,prevenergy) > thrd_Ienergy &&
      abs(currenergy - prevenergy) / max(currenergy,prevenergy) > thrd_T1)
  {
     p_frame_parameters[i_displayorder]->b_scut_loss = true;
     return;
  }
 }
 else
 {
 // Detect by energy difference or motion difference around a scene change
     double thrd_T2 = 0.7;
     double thrd_T3 = 0.7;
     double thrd_rsd_low = 4;
     double thrd_motion_low = 8;
     int Lwin = 10;

     cnt_bef = cnt_aft = 0;
     bef_rsd = aft_rsd = 0;
     bef_motion = aft_motion = 0;
     for (j=1; j< Lwin;j++)
     {
        pos = frmidx -j;
        if (pos < 0)
```

```
            break;
         // Less likely to have two scene changes in a window of 10 frames
         if (p_frame_parameters[pos]->b_scut_loss)
         {
            break;
         }
         if (p_frame_parameters[pos]->i_frame_type == P_frame)
         {
            cnt_bef ++;
            bef_rsd    += p_frame_parameters[pos]->d_residual_var;
            bef_motion += p_frame_parameters[pos]->d_avgmv;
         }
      }
      for (j=1; j< Lwin;j++)
      {
         pos = frmidx +j;
         if (pos == currfrmid)
            break;
         if (p_frame_parameters[pos]->i_frame_type == I_frame ||
            p_frame_parameters[frmidx]->d_IntraRatio >= INTRA_IP)
            break;
         if (p_frame_parameters[pos]->i_frame_type == P_frame)
         {
            cnt_aft ++;
            aft_rsd    += p_frame_parameters[pos]->d_residual_var;
            aft_motion += p_frame_parameters[pos]->d_avgmv;
         }
      }
      if (cnt_bef > 3 && cnt_aft > 3)
      {
         bef_rsd /= cnt_bef;
         bef_motion /= cnt_bef;
         aft_rsd /= cnt_aft;
         aft_motion /= cnt_aft;
         if (max(bef_rsd,aft_rsd) > thrd_rsd_low &&
             abs(bef_rsd - aft_rsd)/max(bef_rsd,aft_rsd) > thrd_T2)
            p_frame_parameters[frmidx]->b_scut_loss = true;
         else if (max(bef_motion, aft_motion) > thrd_motion_low &&
              abs(bef_motion - aft_motion)/max(bef_motion,aft_motion)> thrd_T3)
            p_frame_parameters[frmidx]->b_scut_loss = true;
      }
   }
}
```

3.2.2.3.5 Detect gradual transition picture

When one scene changes gradually to another scene, and if packet loss occurs in a gradual transition picture, the artifacts in the error concealed picture are less visible. This is quite contrary to scene cut artifacts. Even if the scene changes gradually, the energy or motion difference of these scenes may still highly probably be large. Thus, it is better to detect if a scene cut candidate frame is a gradual transition picture. If the scene cut candidate frame is a gradual transition picture, the scene cut artifacts detection algorithm in figure 3-6 will not be applied.

The flow chart for deriving *b_fade_pic* is shown in figure 3-7. The idea is that consecutive frames in a shorter period having larger intra MB ratio than their surrounding frames in a longer period are gradual transition picture positions with higher probability.

For a frame, if its intra-MB ratio is not larger than a lower threshold *INTRA_THRDLOW*, the frame is most probably not a gradual transition picture. Otherwise, check how many consecutive frames having intra-MB ratio larger than *INTRA_THRDLOW*, in the surrounding frames in a short window of *2*winshort* frames' length, and increase the counter *cnt_short* correspondingly. These frames are candidate gradual transition frames. If *cnt_short* is less than a threshold *THD_FADEPICS*, the frame is less likely to be gradual transition picture, because gradual transition content generally takes time to be viewed as fade-in and fade-out pictures. The threshold *THD_FADEPICS* is set to correspond to 0.1 second. And this threshold should not be less than 2 frames; otherwise, the frame is actually a scene cut.

If *cnt_short* is larger than a threshold *THD_FADEPICS*, calculate the average intra-MB ratio of the candidate gradual transition frames. i.e. *fadeintra*. Calculate the average intra-MB ratio, *fadeavg*, of the inter-prediction frames, i.e., P-frames (excluding the candidate gradual transition frames), in the surrounding frames in a longer window of *2*winlong* frames' length. If the ratio *fadeintra/ fadeavg*is larger than a threshold *THD_FADERATIO*, then the current frame is detected as a gradual transition frame.

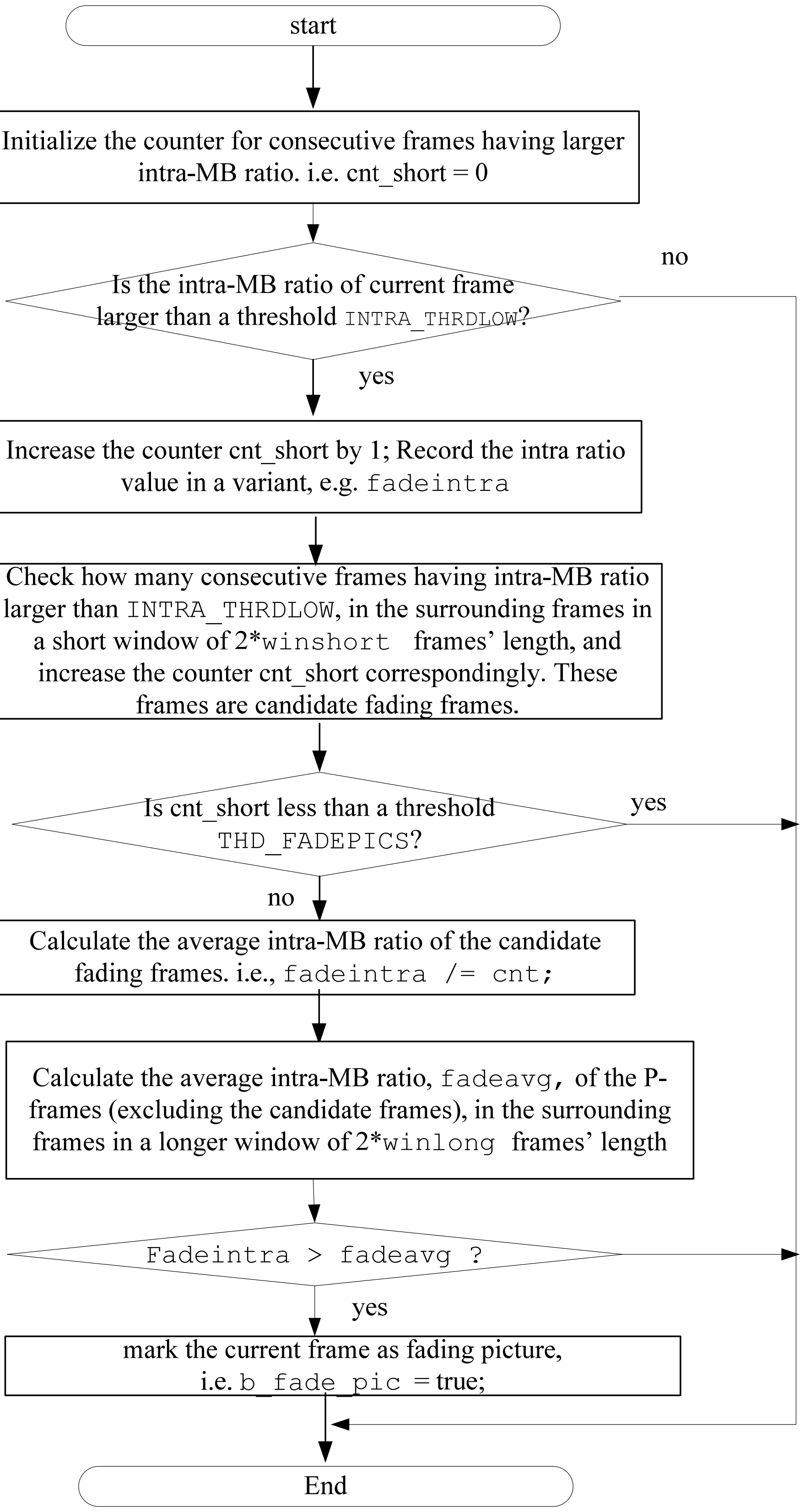

start
Initialize the counter for consecutive frames having larger intra-MB ratio. i.e. cnt_short = 0
Is the intra-MB ratio of current frame larger than a threshold INTRA_THRDLOW?
no
yes
Increase the counter cnt_short by 1; Record the intra ratio value in a variant, e.g. fadeintra
Check how many consecutive frames having intra-MB ratio larger than INTRA_THRDLOW, in the surrounding frames in a short window of 2*winshort frames' length, and increase the counter cnt_short correspondingly. These frames are candidate fading frames.
Is cnt_short less than a threshold THD_FADEPICS?
yes
no
Calculate the average intra-MB ratio of the candidate fading frames. i.e., fadeintra /= cnt;
Calculate the average intra-MB ratio, fadeavg, of the P-frames (excluding the candidate frames), in the surrounding frames in a longer window of 2*winlong frames' length
Fadeintra > fadeavg ?
yes
mark the current frame as fading picture, i.e. b_fade_pic = true;
End

Figure 3-7 - Flow chart of detecting a gradual transition picture

The codes to derive b_fade_pic is:

```
void detect_transitionPicture(int frmidx, int currfrmid)
{
   int winshort = f_fps * 0.5;
   int winlong = f_fps * 1.25;
   double INTRA_THRDLOW = 0.4;
   int THD_FADEPICS = max(2,( f_fps * 0.1 + 0.5));
   double THD_DIVFRMES =  f_fps *1.0;
   double THD_FADERATIO = 3;
   float fadeavg,fadeintra;
   int cnt, pos, pos_bef, pos_aft,i;

   /* consecutive frames having larger intra ratio are potential
   fade-in and fade-out position */
   fadeintra = 0; cnt = 0;
   if (p_frame_parameters[frmidx]->d_IntraRatio > INTRA_THRDLOW)
   {
      fadeintra += p_frame_parameters[frmidx]->d_IntraRatio; cnt ++;
   }
   else
      return;

   for (i= 1; i< winshort; i++)
   {
       pos = frmidx - i;
       if (pos <= 0 || p_frame_parameters[pos]->d_IntraRatio < INTRA_THRDLOW)
          break;
      if (p_frame_parameters[pos]->i_frame_type == I_frame)
       {
          fadeintra += 0.6; cnt ++;
       }
      else if (p_frame_parameters[pos]->i_frame_type == P_frame)
       {
         fadeintra += p_frame_parameters[pos]->d_IntraRatio; cnt ++;
       }
   }
   pos_bef = i;
   for (i= 1; i<winshort; i++)
   {
      pos = frmidx + i;
      if (pos == currfrmid ||p_frame_parameters[pos]->d_IntraRatio <
             INTRA_THRDLOW)
         break;
      if (p_frame_parameters [pos]->i_frame_type == I_frame)
      {
         fadeintra += 0.6; cnt ++;
      }
      else if (p_frame_parameters[pos]->i_frame_type == P_frame)
      {
         fadeintra += p_frame_parameters[pos]->d_IntraRatio; cnt ++;
      }
   }
   pos_aft = i;

   if (cnt < THD_FADEPICS)
         return;
   fadeintra /= cnt;

/*the average intra MB ratio of potential fading frames should be much larger
than the average intra MB ratio in the long window */
```

```
   fadeavg = 0; cnt = 0;
   for (i= pos_bef; i<winlong; i++)
   {
      pos = frmidx - i;
      if (pos <= 0)
         break;
      if (p_frame_parameters[pos]->i_frame_type == P_frame)
      {
         fadeavg += p_frame_parameters[pos]->d_IntraRatio; cnt ++;
      }
   }
   for (i= pos_aft; i<winlong; i++)
   {
      pos = frmidx + i;
      if (pos == i_processed_frames)
         break;
      if (p_frame_parameters[pos]->i_frame_type == P_frame)
      {
         fadeavg += p_frame_parameters[pos]->d_IntraRatio; cnt ++;
      }
   }
    if (cnt > THD_DIVFRMES)
    {
       fadeavg /= cnt;
       if (fadeintra > fadeavg * THD_FADERATIO)
          p_frame_parameters[frmidx]->b_fade_pic = true;
    }
}
```

3.2.3 Freezing module parameters

```
Input  = [i_frame_type,
          f_fps,
          p_mb_data_array[]]

Output = [p_frame_parameters[]->d_pan_factor,
          p_frame_parameters[]->d_zoom_factor]
```

3.2.3.1 Input parameter description

i_frame_type	See subsection 3.1.3.2
f_fps	See subsection 3.1.1.2
p_mb_data_array[]	See subsection 3.1.3.1

3.2.3.2 Output parameter description

p_frame_parameters[] Array holding parameters for each parsed picture.

The following output parameters in *p_frame_parameters[]* are calculated for the inter-predicted frame immediately before a freeze event:

d_pan_factor Quantifies the isotropic homogeneity of motion in a frame. The initial value is zero.

d_zoom_factor	Quantifies the radial homogeneity of motion in a frame. The initial value is zero.

3.2.3.3 Algorithms

d_pan_factor quantifies the isotropic homogeneity like panning while *d_zoom_factor* specifies the radial homogeneity like zooming.

For an inter-predicted picture before a freezing event, *d_pan_factor* is defined as the magnitude of the vector mean of all MVs in the picture as:

$$d\_pan\_factor = \frac{1}{i\_nbr\_mbs}\sqrt{\left(\sum_{r\in\tau}\frac{\sum_{l\subset r} MV_{\mathrm{h},l,r}\cdot A_{l,r}}{\sum_{l\subset r} A_{l,r}}\right)^2 + \left(\sum_{r\in\tau}\frac{\sum_{l\subset r} MV_{\mathrm{v},l,r}\cdot A_{l,r}}{\sum_{l\subset r} A_{l,r}}\right)^2}$$

where r indexes the MBs in the τ-th picture and l indexes the partitions in the r-th MB; $MV_{\mathrm{h},l,r}$ and $MV_{\mathrm{v},l,r}$ and $A_{l,r}$ denote the horizontal MV, vertical MV and area (number of pixels) of the l-th partition in the r-th MB respectively; note that all the MVs have been clipped within the range of [-128, 128] and then multiplied with the frame rate of the video so they depend on only the content motion but not the video format; i_nbr_mbs is the number of the macro blocks in a picture. *d_pan_factor* is greater when a big foreground in the scene is translating or the camera is panning.

For an inter-predicted picture before a freezing event, *d_zoom_factor* is more complicated to compute. Firstly, the difference between the sum of horizontal MVs in the left half picture and those in the right half picture, and the difference between the sum of vertical MVs in the top half picture and those in the bottom half picture are both calculated. Secondly, the two difference values are both normalized by the total number of the macro blocks in a picture, i.e., i_nbr_mbs, and form a 2D vector. Thirdly, *d_zoom_factor* is set as the magnitude of the vector.

$$d\_zoom\_factor = \frac{1}{\mathrm{i\_nbr\_mbs}} \times$$

$$\sqrt{\left|\sum_{r\in\tau_{\mathrm{L}}}\frac{\sum_{l\subset r} MV_{\mathrm{h},l.r}A_{l,r}}{\sum_{l\subset r} A_{l,r}} - \sum_{r\in\tau_{\mathrm{R}}}\frac{\sum_{l\subset r} MV_{\mathrm{h},l.r}A_{l.r}}{\sum_{l\subset r} A_{l,r}}\right|^2 + \left|\sum_{r\in\tau_{\mathrm{T}}}\frac{\sum_{l\subset r} MV_{\mathrm{v},l.r}A_{l,r}}{\sum_{l\subset r} A_{l,r}} - \sum_{r\in\tau_{\mathrm{B}}}\frac{\sum_{l\subset r} MV_{\mathrm{v},l.r}A_{l,r}}{\sum_{l\subset r} A_{l,r}}\right|^2}$$

where τ_{L}, τ_{R}, τ_{T} and τ_{B} represent the left, right, top and bottom half plane of the τ-th picture respectively. *d_zoom_factor* is great if the camera is zooming, since *d_zoom_factor* is cumulated over the radial symmetric motion vectors.

3.3 Aggregation of parameters into model parameters

This subsection describes how the aggregation of parameters into model parameters is made for each of the degradation modules.

3.3.1 Compression module parameters

```
Input  = [i_total_slice_qp,
          i_nbr_total_slice_qp,
          f_frame_content_complexity[],
          i_nbr_error_free_intra_frame]


Output = [f_video_qp,

          f_video_content_complexity]
```

3.3.1.1 Input parameter description

i_total_slice_qp	See subsection 3.2.1.2
i_nbr_total_slice_qp	See subsection 3.2.1.2
f_frame_content_complexity[]	See subsection 3.2.1.2
i_nbr_error_free_intra_frame	See subsection 3.2.1.2

3.3.1.2 Output parameter description

f_video_qp	Video quantization parameter of video sequence.
f_video_content_complexity	Video content complexity of video sequence. Its default value is 30.0.

3.3.1.3 Algorithms

3.3.1.3.1 Obtain video quantization parameter

The parameter, *f_video_qp*, is the average value of all *i_slice_qp* of the sequence and is calculated as below:

$$f\_video\_qp = \frac{\text{i\_total\_slice\_qp}}{\text{i\_nbr\_total\_slice\_qp}}$$

where *f_video_qp* is the video quantization parameter of the video sequence.

3.3.1.3.2 Obtain video content complexity

The parameter, *f_video_content_complexity*, is the average value of frame content complexity of all *error free* Intra frame (i.e. *f_frame_content_complexity[]*) of the sequence and is calculated as below:

$$f\_video\_content\_complexity = \frac{\sum_{\text{j}=1}^{\text{i\_nbr\_error\_free\_intra\_frame}} f\_frame\_content\_complexity_{\text{j}}}{\text{i\_nbr\_error\_free\_intra\_frame}}$$

where j is from 1 to the *i_nbr_error_free_intra_frame*, inclusive; $f\_frame\_content\_complexity_{\text{j}}$ means the frame content complexity of the j-th *error free* Intra frame; *f_video_content_complexity* is the video content complexity of the video sequence.

For implementation, pseudo code is provided below:

```
float f_total_frame_content_complexity = 0.0f;
for( i = 0; i < i_nbr_error_free_intra_frame; i++ )
{
        f_total_frame_content_complexity += f_frame_content_complexity[i];
```

}

f_video_content_complexity = f_total_frame_content_complexity / i_nbr_error_free_intra_frame;

3.3.2 Slicing module parameters

```
Input  = [i_processed_frames,
          i_nbr_mbs,
          p_frame_parameters[],
          f_fps]

Output = [d_LoVA_seq]
```

3.3.2.1 Input parameter description

i_processed_frames	See subsection 3.2.2.2
i_nbr_mbs	See subsection 3.1.2.2
p_frame_parameters[]	See subsection 3.2.2.2
f_fps	See subsection 3.1.1.2

3.3.2.2 Output parameter description

d_LoVA_seq	Measurement of level of visible slicing artifacts exist in a sequence of pictures. The value is a non-negative number. A value of 0 means there is no visible slicing artifact. If *s_video_PLC_mode* is not "SLICING", this value should be 0.

3.3.2.3 Algorithms

3.3.2.3.1 Estimate frame-level visible artifact level

Generally, the goal of error concealment is to estimate EC MBs in order to minimize perceptual quality degradation. The perceived strength of artifacts produced by transmission errors depends heavily on the employed error concealment techniques. For example, if a frame far away from a current frame is used to conceal a current macro block, the concealed macro block is more likely to have visible artifacts. In addition, the artifact strength is also related to the video content. For example, a slow moving video is easier to be concealed. Thus, parameters, such as motion vectors and error concealment distance, can be used to assess the error concealment effectiveness and the quality of concealed video at a bitstream level.

Visual artifacts may still be perceived after error concealment, because error concealment may be not effective therein. Such visual artifacts caused by EC MB are denoted as initial visible artifacts. If a block having initial visible artifacts is used as a reference, for example, for intra prediction or inter prediction, the initial visible artifacts may propagate spatially or temporally to other macro blocks in the same or other frames through prediction. Such propagated artifacts are denoted as propagated visible artifacts. The overall artifacts, caused by initial and/or propagated visible artifacts, are denoted as overall visible artifacts.

For a MB indexed as (i,j) of frame n, its initial level of visible artifacts *d_lova* is set as

$$LoVA_{init}^{curr}(n,i,j) = \begin{cases} 0, & \text{nonEC MB} \\ f(d\_mvmedian_{nij} * ecdist/4.0), & \text{EC MB} \end{cases} \quad (1)$$

$$f(x) = \begin{cases} v_1, & ,x < \mathrm{T}_1 \\ \frac{(v_2 - v_1)}{\mathrm{T}_2 - \mathrm{T}_1} * (x - \mathrm{T}_1) & ,\mathrm{T}_1 \le x \le \mathrm{T}_2 \\ v_2 & ,x > \mathrm{T}_2 \end{cases} \quad (2)$$

The reason for the denominator 4.0 in the above equation is that the motion estimation accuracy in H.264 is ¼ sub-pixel. The constants take following values: $\mathrm{v}_1 = 0$, $\mathrm{v}_2 = 100$; $\mathrm{T}_1 = 1$ and $\mathrm{T}_2 = 8$ in the unit of pixel.

If its reference frame is lost completely, i.e., *b_reflost0 == true*, or *b_reflost1 == true*, additional initial artifacts caused by lost reference frame is calculated for correctly received inter-predicted MB:

$$LoVA_{init}^{ref}(n,i,j) = f(d\_mvmedian\_ref * ecdist/4.0) \quad (3)$$

where the magnitude of the median of its motion vectors pointed to the lost reference frame is calculated as $d\_mvmedian\_ref$. When there is no corresponding motion vector for the macro block, $d\_mvmedian\_ref$ is set to zero. Since Bi-directional prediction may be used, $d\_mvmedian\_ref0$ is the value corresponding to the lost forward reference frame, and $d\_mvmedian\_ref1$ is the value corresponding to the lost backward reference frame. Otherwise (i.e. b_reflost0 == false && b_reflost1 == false), $\mathrm{LoVA}_{\mathrm{init}}^{\mathrm{ref}}(\mathrm{n,i,j}) = 0$.

Considering both lost macro blocks in the current frame and the loss of reference frames, the initial visible artifact level is calculated as:

$$\mathrm{LoVA_{init}(n,i,j)} = \max\{\mathrm{LoVA_{init}^{curr}(n,i,j)}, \mathrm{LoVA_{init}^{ref0}(n,i,j)}, \mathrm{LoVA_{init}^{ref1}(n,i,j)}\} \quad (4)$$

The overall artifacts in a MB should consider both initial artifacts and propagated artifacts. The artifact level of a pixel $\mathrm{d(n,x,y)}$, where n is the frame index, (x,y) is the pixel's coordinate in the frame, is calculated as follows:

$$\mathrm{d(n,x,y)} = \max\left(\mathrm{d(n-k,x',y')},\ \mathrm{LoVA_{init}(n,x/16,y/16)}\right),$$

where $\mathrm{d(n-k,x',y')}$is the propagated visible artifact for pixel (n,x,y), and is the artifact level at reference pixel (n-k,x',y') of the current pixel. The initial value of $\mathrm{d(n,x,y)}$ is zero, meaning no visible artifact. In summary, how the artifact level propagates may be traced through motion vectors. In this implementation, it is sufficient to use zero motion vectors instead of pixel-accurate motion vectors to roughly track the temporal propagation of visible artifacts, i.e., it is sufficient to simply work at the MB level and the distortion of all the pixels $\mathrm{d(n,x,y)}$, in a MB index by (n,i,j) is of the same value denoted as $\mathrm{d\_lova(n,i,j)}$, that is:

$$\mathrm{d\_lova(n,i,j)} = \max(\mathrm{d\_lova(n-k,i,j)}, \mathrm{LoVA_{init}(n,i,j)})$$

(5)The initial value of $\mathrm{d\_lova(n,i,j)}$ is zero, meaning no visible artifact.

In case that the frame having scene cut artifact, i.e., *b_scut_loss == true*, the above method to estimate visible artifacts of an EC MB does not work well, because an assumption is made when estimate the *d_mvmedian* of the EC MB. The assumption is that the current picture and the previous pictures are highly correlated, thus the motion and residual in these frames are similar. When scenes change, this assumption does not hold. Generally, the scene cut artifact is very strong. So, when a frame is detected as having scene cut artefact, modify the macro block's visible artifact level to a larger value, as:

$$\mathrm{d\_lova(n,i,j)} = \max(\mathrm{d\_lova(n,i,j)},\ 100) \quad (6)$$

The visible artifact level of a frame *n* is calculated as the weighted sum of all MBs' d_lova as shown in section 3.3.2.3.2.

For the frame that is lost completely (i.e. *p_frame_parameters[n]->i_frame_type == UNKNOWN*), set its *d_LoVA(n)* to that of its previous frame.

3.3.2.3.2 Weight frame-level artifacts

The artifact level for a macro block in a frame is weighted by a vision sensitivity factor to more accurately estimate the perceived artifact strength. The artifacts level of each macro block in a frame is weighted according to its distance to the center of the frame. The idea is that the artifacts located near the center of a frame are more annoying than those located near the edge of the frame.

$$\mathrm{d\_LoVA}(n) = \sum_{i=0}^{\mathrm{i\_nbr\_mb\_ver}} \sum_{j=0}^{\mathrm{i\_nbr\_mb\_hor}} \mathrm{d\_lova}(n,i,j) \times \left(1 - \frac{\sqrt{\left(i - \frac{\mathrm{i\_nbr\_mb\_ver}}{2}\right)^2 + \left(j - \frac{\mathrm{i\_nbr\_mb\_hor}}{2}\right)^2}}{\sqrt{\left(\frac{\mathrm{i\_nbr\_mb\_ver}}{2}\right)^2 + \left(\frac{\mathrm{i\_nbr\_mb\_hor}}{2}\right)^2}}\right) \qquad (7)$$

The pseudo codes for deriving the initial visible artifact estimation, propagated artifact and overall visible artifact level of the MBs in a frame, d_lova, are:

```
void EC_EP_artifacts (frmidx)
{
  int MAX_ARTIFACT = 100;

   struct MB_parameters * currfrm_mbparameter =
                    p_frame_parameters[frmidx]->p_mb_parameters;
   struct MB_parameters * prevfrm_mbparameter, * backfrm_mbparameter;

  // Estimate initial visible artifacts for EC MB according to Eq. (1)
  if (p_frame_parameters[frmidx]->i_nbr_ecmbs > 0)
  {
    for (int i=0;i<i_nbr_mbs;i++)
    {
       if (currfrm_mbparameter[i].b_ec)
       {
         int judgement1 = currfrm_mbparameter[i].d_mvmedian /4.0 *
                p_frame_parameters[frmidx]->i_ecdist;
         artifacts_level = f(judgement1); // refer to equation (2)

         currfrm_mbparameter[i].d_lova = artifacts_level;
       }
    }
  }

  // Additional initial artifacts caused by lost reference frame per Eq. (3)
  if (p_frame_parameters[frmidx]->b_reflost0)
  {
    for (int i=0;i<i_nbr_mbs;i++)
    {
      if (!currfrm_mbparameter[i].b_intramode &&
          currfrm_mbparameter[i].b_ec == false)
      {
         int judgement1 = abs(currfrm_mbparameter[i].d_mvmedian_ref0 /4.0 *
                p_frame_parameters[frmidx]->i_ecdist);
         artifacts_level = f(judgement1); // refer to equation (2)

         // Count in the effects of both types of loss according to Eq. (4)
         currfrm_mbparameter[i].d_lova =
                max(artifacts_level, currfrm_mbparameter[i].d_lova);
      }
    }
  }

  if (p_frame_parameters[frmidx]->b_reflost1)
  {
    for (int i=0;i<i_nbr_mbs;i++)
    {
      if (!currfrm_mbparameter[i].b_intramode &&
          currfrm_mbparameter[i].b_ec == false)
      {
        int judgement1 = abs(currfrm_mbparameter[i].d_mvmedian_ref1 /4.0 *
                p_frame_parameters[frmidx]->i_ecdist);
          artifacts_level = f(judgement1); // refer to equation (2)

        // Count in the effects of both types of loss according to Eq. (4)
        currfrm_mbparameter[i].d_lova =
                max(artifacts_level, currfrm_mbparameter[i].d_lova);
      }
    }
  }

  // In case that the frame having scene cut artifact, modify the MB's
  // visible artifact level to a larger value
  if (p_frame_parameters[frmidx]->b_scut_loss == true)
  {
    for (int i=0;i<i_nbr_mbs;i++)
    {
      if (currfrm_mbparameter[i].b_ec || (p_frame_parameters[frmidx]->b_reflost0
          && lova_frames[k]->frame_type == P_frame))
       currfrm_mbparameter[i].d_lova =
          max(currfrm_mbparameter[i].d_lova, MAX_ARTIFACT);
    }
  }

  // Track visible artifact propagation according to Eq. (5),
  // and stop artifact propagation when an Intra MB is correctly received
  for (int i=0;i<i_=nbr_mbs;i++)  {
    if (currfrm_mbparameter[i].b_intramode)
      currfrm_mbparameter[i].d_lova = 0;
    else
    {
       if (currfrm_mbparameter[i].c_predDirection < 0x02)
       {
        prevfrm_mbparameter=p_frame_parameters[p_frame_parameters[frmidx]->
          i_refFramesIdx[0]]->p_mb_parameters;
        currfrm_mbparameter[i].d_lova =
           max(currfrm_mbparameter[i].d_lova,prevfrm_mbparameters[i].d_lova);
       }
       else if (currfrm_mbparameter[i].c_predDirection == 0x02)
       {
        backfrm_mbparameter=p_frame_parameters[p_frame_parameters[frmidx]->
          i_refFramesIdx[1]]->p_mb_parameters;
        currfrm_mbparameter[i].d_lova =
           max(currfrm_mbparameter[i].d_lova,backfrm_mbparameters[i].d_lova);
       }
       else
       {
        prevfrm_mbparameter=p_frame_parameters[p_frame_parameters[frmidx]->
          i_refFramesIdx[0]]->p_mb_parameters;
        backfrm_mbparameter=p_frame_parameters[p_frame_parameters[frmidx]->
```

```
            i_refFramesIdx[1]]->p_mb_parameters;
          currfrm_mbparameter[i].d_lova =
              max(currfrm_mbparameter[i].d_lova,(prevfrm_mbparameters[i].d_lova +
             backfrm_mbparameters[i].d_lova)/2);
        }
  }
}
```

3.3.2.3.3 Obtain sequence-level visible artefact level

In the above section, the spatial artifact level of a frame is estimated. We use "spatial artifact" to denote artifact perceived in a picture in a video sequence when the picture is viewed independently of other pictures in the video sequence, and use "temporal artifact" to denote artifact that is perceived in a picture of a video sequence when pictures in the video sequence are continuously displayed. Spatial artifact in pictures needs to last for a period of time so that eyes can fix on and recognize it as artifact. When the pictures are part of a video sequence and each is displayed only for a very short period of time (for example, a period of 1/f_fps when the video is played in real time), the perceived video distortion at the time instant of frame n, i.e., temporal distortion at frame n, d_CLoVA(n), can be quite different from spatial distortion of frame n, d_LoVA(n). The perceived temporal distortion at time instant of frame n depends on the context of its neighboring frames. The context includes the duration and the pattern of the distortion.

In a neighborhood around current frame n, find a sliding window of T_0 duration, which the current frame belongs to and has a highest density of *large distortion* (i.e., the distortion level exceeds a certain threshold, U). Then in the identified sliding window, the ratio between the number of frames with large distortion and the total number of frames is used as a context feature to weight current frame's spatial distortion d_LoVA(n). The flow chart is shown in figure 3-8.

Specifically, As shown in the first step in figure 3-8, a sliding window of L0 frames that includes frame n (denoted as $S_{i,n}$) starts at frame $(n - i)$ and ends at frame $(n - i + L_0 - 1)$, $0 \le i < L_0$. For each sliding window $S_{i,n}$, obtain the ratio, $R_{i,n}$, between the number of frames with large distortion in $S_{i,n}$ and the total number of frames in $S_{i,n}$ by

$$R_{i,n} = \frac{\sum_j \mu(d\_LoVA(j))}{L_0}\ ,\quad \text{frame } j \in S_{i,n} \tag{8}$$

where $\mu(x) = \begin{cases} 1, & x \ge U \\ 0, & \text{otherwise} \end{cases}$, U is the lower-bound threshold of image artifacts and set to i_nbr_mbs/100 in this implementation. We denote the ratio between the number of frames with large spatial distortion in a sliding window and the total number of frames in the sliding window as a *large distortion density,* i.e. $R_{i,n}$, for the sliding window.

Further, identify the window having the highest *large distortion density* among all the sliding windows that include current frame n. The *large distortion density* of the identified window is:

$$w_n = \max\{R_{i,n}\ ,\ 0 \le i < L_0\} \tag{9}$$

Another feature is the distance between frame n and the closest frame with *large distortion* in the sliding window W_n corresponding to the highest *large distortion density*, i_dist. If there is no other frame with *large distortion* in the corresponding sliding window, leave dist(n) to the default value, a very big value of 1000. The idea is that, when two frames with *large distortion* are closer, the distortion becomes more visible to human eyes.

The perceived video distortion at frame n is calculated as the weighted distortion:

$$d\_CLoVA(n) = w_n \times d\_LoVA(n)/i\_dist_n \tag{10}$$

The sequence-level artifact is calculated as:

$$d\_LoVA\_seq = \log10(\sum_{n=1}^{i\_processed\_frames} d\_CLoVA(n))/f\_fps + 1) \qquad (11)$$

This metric d_LoVA_seq will be used as slicing module output to the framework algorithm.

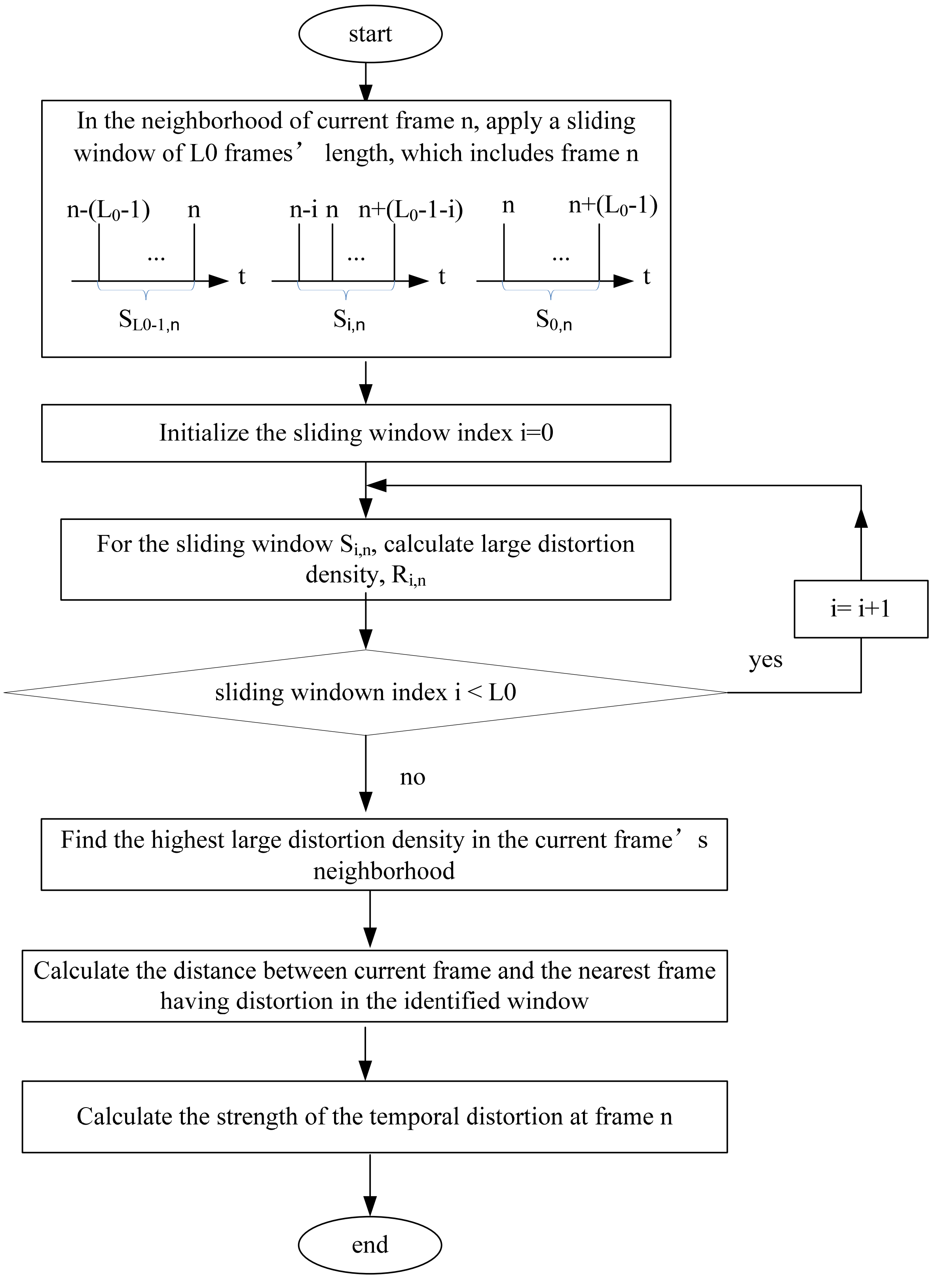

start
In the neighborhood of current frame n, apply a sliding window of L0 frames' length, which includes frame n
n-(L0-1)
n
...
t
SL0-1,n
n-i n
n+(L0-1-i)
Si,n
n
n+(L0-1)
S0,n
Initialize the sliding window index i=0
For the sliding window Si,n, calculate large distortion density, Ri,n
sliding windown index i < L0
yes
i= i+1
no
Find the highest large distortion density in the current frame's neighborhood
Calculate the distance between current frame and the nearest frame having distortion in the identified window
Calculate the strength of the temporal distortion at frame n
end

Figure 3-8 - Flow diagram depicting the method for modeling temporal distortion at frame n

3.3.3 Freezing module parameters

```
Input  = [i_total_num_freezing_frames,
          i_total_num_frames,
          p_frame_parameters[]->d_pan_factor,
          p_frame_parameters[]->d_zoom_factor]

Output = [f_freezing_ratio,
          d_MV]
```

3.3.3.1 Input parameter description

i_total_num_freezing_frames	The number of freezing frames for the video sequence, i.e. sum of all freezing frames of all freezing events.
i_total_num_frames	The total number of frames for the video sequence.
d_pan_factor	See subsection 3.2.3.2
d_zoom_factor	See subsection 3.2.3.2

3.3.3.2 Output parameter description

f_freezing_ratio	ratio of i_total_num_freezing_frames and i_total_num_frames.
d_MV	quantifies the motion homogeneity.

3.3.3.3 Algorithms

If the video contains *a pause, also referred to as a freeze, with skipping,* the distorted video contains a group of consecutive frames that are dropped. Every freeze with skipping (shorted as freeze) includes one or more consecutive pictures, which are dropped and replaced by the last correctly received picture for the displayed video.

f_freezing_ratio, the freezing feature, is the ratio of i_total_num_freezing_frames and i_total_num_frames:

$$\text{f\_freezing\_ratio} = \frac{\text{i\_total\_num\_freezing\_frames}}{\text{i\_total\_num\_frames}}$$

d_MV, the motion feature, quantifies the motion homogeneity:

$$d\_MV = \frac{1}{\mathrm{N}}\sum_{i=1}^{N} \max\{d\_pan\_factor_i, d\_zoom\_factor_i\}$$

where *N* is the total number of the freezing events. *d_pan_factor*$_i$ and *d_zoom_factor*$_i$ are respectively the *d_pan_factor* and the *d_zoom_factor* of the inter-predicted picture just before the *i*-th freezing event. For the *i*-th pause, the greater value between *d_pan_factor*$_i$ and *d_zoom_factor*$_i$ is selected. *d_MV* is defined as the average greater values over the total of *N* freezing events.

3.4 Quality estimation model

The quality estimation model consists of quality/artifact estimation modules for compression, slicing, freezing with skipping and a framework module that is used to combine the various degradations into a total video quality score. When artifact is only caused by video compression, the *d_compression_quality_value* is output directly as the overall video quality score. Otherwise, when there is one or more other artifact type, for example, artifact caused by packet losses, the framework module is used to consider degradations caused by various artifact types to generate a total video quality score.

3.4.1 Compression module

The compression quality estimation module is for estimating the video quality due to compression and it has the following input and output parameters:

```
Input  = [f_video_qp,
          f_video_content_complexity]
Output = [d_compression_quality_value]
```

3.4.1.1 Input parameter description

f_video_qp	See subsection 3.1.3.2
f_video_content_complexity	See subsection 3.2.1.2

3.4.1.2 Output parameter description

d_compression_quality_value	Estimation of the compression quality for video sequence

3.4.1.3 Coefficients

```
For SD resolution
c1 = 1.4163
c2 = 2.9116
c3 = 1.0
c4 = 41.5
c5 = 4.7
c6 = 13.0

For 1280x720 resolution
c1 = 1.0519
c2 = 3.3876
c3 = 1.0
c4 = 40.0
c5 = 0.75
c6 = 10.0
```

```
For 1920x1080 resolution interlace (1080i)
c1 = 1.2294
c2 = 3.1092
c3 = 1.0
c4 = 41.5
c5 = 0.65
c6 = 10.5

For 1920x1080 resolution progressive (1080p)
c1 = 1.2294
c2 = 3.1092
c3 = 1.0
c4 = 43.0
c5 = 0.85
c6 = 12.0
```

3.4.1.4 Algorithm

f_video_content_complexity is normalized as below:

$$f\_video\_content\_complexity = \min\left(\text{n1}, \sqrt{\frac{f\_video\_content\_complexity}{\text{n2}}}\right)$$

where n1 is 1.0, n2 is 60.0

d_compression_quality_value is calculated as below:

$$\text{d\_compression\_quality\_value} = \text{c}_1 + \frac{\text{c}_2}{\text{c}_3 + \left(\frac{f\_video\_qp}{\text{c}_4 - \text{c}_5 * f\_video\_content\_complexity}\right)^{\text{c}_6}}$$

where $c_{1,}$ c_2 $c_{3,}$ c_4 $c_{5,}$ and c_6 are coefficients and as shown in section 3.4.1.3

3.4.2 Slicing module

The following inputs and outputs are used when calculating a slicing artifact value indicative of the visible degradation due to slicing:

```
Input  = [d_LoVA_seq]

Output = [d_slicing_artifact_value]
```

3.4.2.1 Input parameter description

d_LoVA_seq	See subsection 3.3.2.2

3.4.2.2 Output parameter description

d_slicing_artifact_value	A float value indicating the estimated level of slicing artifact for the sequence.

3.4.2.4 Algorithm

The overall slicing artifact value is derived as:

```
d_slicing_artifact_value = d_LoVA_seq;
```

3.4.3 Freezing module

The following inputs and outputs are used when calculating a freezing artefact value indicative of the visible degradation due to freezing:

```
Input =  [f_freezing_ratio,
          d_MV,
          f_fps]

Output = [d_freezing_artifact_value]
```

3.4.3.1 Input parameter description

f_freezing_ratio	See subsection 3.3.3.2
d_MV	See subsection 3.3.3.2
f_fps	See subsection 3.1.1.2

3.4.3.2 Output parameter description

d_freezing_artifact_value	Estimation level of the freezing artifact for the sequence. This value should be 0.0 when the video sequence is *error free* or s_video_PLC_mode is not "FREEZING".

3.4.3.3 Coefficients

For SD resolution:

f_1 = 4.773819

f_2 = 0.725262

f_3 = 0.089219

For 720p resolution:

f_1 = 7.411672

f_2 = 0.914548

f_3 = 0.066144

For HD1080 resolution:

f_1 = 3.236362

f_2 = 0.758998

f_3 = 0.064108

3.4.3.4 Algorithms

The *d_freezing_artifact_value* is related to *f_fps* (i.e frame rate), *f_freezing_ratio* (i.e. freezing feature) and *d_MV* (i.e. motion feature), the estimated freezing degradation is calculated according to the following formula:

$$\text{d\_freezing\_artifact\_value} = \frac{4}{1 + \frac{f_1}{\text{f\_fps} * \text{f\_freezing\_ratio}^{f_2} * \text{d\_MV}^{f_3}}}$$

where f_1, f_2 and f_3 are coefficients and as shown in section 3.4.3.3.

3.4.4 Framework for combining modules

The framework for combining modules has the following input and output parameters.

```
Input  = [d_compression_quality_value,
          d_slicing_artifact_value,
          d_freezing_artifact_value]


Output = [d_combined_quality_value]
```

3.4.4.1 Input parameter description

d_compression_quality_value	See subsection 3.4.1.2
d_slicing_artifact_value	See subsection 3.4.2.2
d_freezing_artifact_value	See subsection 3.4.3.2

3.4.4.2 Output parameter description

d_combined_quality_value	A float number between 1 and 5 indicating the level of video quality, considering all kinds of artifacts, i.e. estimated total video MOS.

3.4.4.3 Coefficients

For SD resolution:

```
d_alpha   = [1.0471,    0.0229,   -0.6302]
d_beta2   = [4.0864,    5.2781]
```

For 720p resolution:

```
d_alpha   = [0.9545,    0.1229,   -0.5099]
d_beta2   = [3.7298,    6.0000]
```

For HD1080 resolution:

```
d_alpha   = [0.9109,    0.1533,   -0.5597]
d_beta2   = [3.8509,    5.9577]
```

3.4.4.4 Algorithms

d_compression_quality_value, *d_slicing_artifact_value*, and *d_freezing_artifact_value* are provided by the above sub-sections respectively. The general procedure for combining the three degradation types into a predicted quality MOS is as follows.

First, the estimated levels of distortion for different artifact types are aligned, that is, the estimated level of artifact is mapped to the same quality level, regardless of the artifact type. Specifically, all artifact values are aligned to the ACR MOS scale using specific alignment functions. Slicing is aligned using a negative exponential function. Freezing is aligned using a linear function:

$$f = 5 - x, \qquad 0 \le x \le 4$$

Second, identify the dominant visual artifacts by sorting the aligned quality values, *d_compression_quality_value*, *d_slicing_quality_value*, and *d_freezing_quality_value* in ascending order. The sorted results are stored in the array *dpp[0...2]*. Therefore, *dpp[0]* is the estimated level for the artifact with highest quality impact while *dpp[2]* is the estimated level for the artifact with lowest quality impact.

Third, linearly combine the dominant visual artifacts into a predicted quality MOS. Human perception will be more influenced by the most significant artifact with value *dpp[0]*, then *dpp[1]*. Linear combination is applied to get the overall quality value from *dpp[0]* and *dpp[1]*:

$$d\_combined\_quality\_value = \alpha_1 \cdot dpp[0] + \alpha_2 \cdot {*}dpp[1] + \alpha_3$$

wherein $|\alpha_1| > |\alpha_2|$.

The combined quality value is then estimated according to the following pseudocode:

```
d[0] = d_compression_quality_value
d[1] = d_slicing_artifact_value
d[2] = d_freezing_artifact_value

// Align to MOS
dp[0] = d[0]
dp[1] = -exp(d[1]/d_beta2[0]) + d_beta2[1]
if (d[1]==0) dp[1] = 5;
dp[2] = 5 - d[2]

dpp = sorting(dp)          // sorting in ascending order

// Combine the two most significant artifacts with a linear function
d_combined_quality_value = d_alpha[0]*dpp[0] + d_alpha[1]*dpp[1] + d_alpha[2]

// Make sure scores are between 1 and 5
d_combined_quality_value = min(max(d_combined_quality_value, 1.0), 5.0)
```

4 Model mode 2 description

Recommendation ITU-T P.1202.2 Mode 2 has a concise mathematical form, and yet the partial factors in the model may involve complex steps of feature extraction. The following description is based on texts, formula and diagrams, while the reader is referred to JSON descriptions for the data structure, the pseudo-code for the detailed algorithm, and the paper [Zhang, et.al., 2013] for the design considerations.

Figure 4-1 outlines the high-level stages of Recommendation ITU-T P.1202.2 Mode 2 algorithm.

Figure 4-2 presents the flow of parameters in the model

Figure 4-3 sketches the process of feature extraction.

Figure 4-4 depicts the steps of slicing key-factor extraction.

Figure 4-5 and 4-6 illustrate the detailed algorithms for calculating special features.

This section describes the Recommendation ITU-T P.1202.2 model mode 2 and how it is implemented. The block diagram for the model is depicted in figure 4-1. The model takes an H.264/AVC baseline encoded video bitstream and side information (e.g. error concealment type,) as input, extracts parameters, and aggregates them into sequence-level features which are used to calculate an estimated video quality MOS for the sequence.

The model description is organized according to the flow of parameters in the model and to what degradation module the parameters belong to. In the Subsection 4.1, the algorithms for aggregating the basic parameters into features at sequence level are described, and finally the quality is predicted according to the Eq. 4-1.

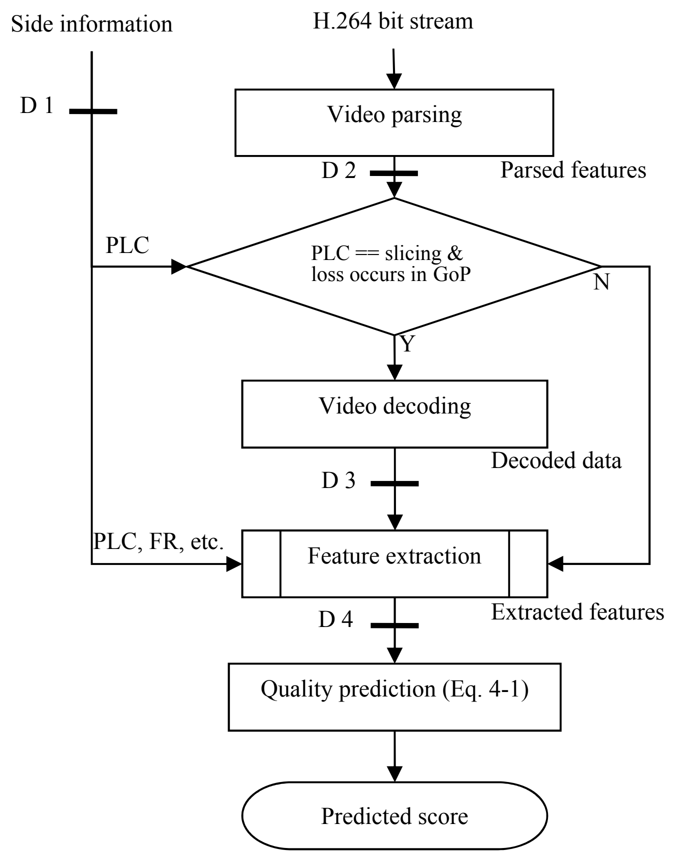


Figure 4-1 – General overview of the ITU-T P.1202.2 Mode 2 algorithm

Three different types of degradations are detected; compression artifacts (c), freezing artifacts (f), and slicing artifacts (s). Compression artifacts are introduced due to lossy compression during the encoding process. Slicing artifacts are introduced when packet losses are concealed using a packet loss concealment (PLC) scheme trying to repair erroneous frames. Freezing artifacts are introduced when the PLC scheme of the receiver replaces the erroneous frames (either due to packet loss or error propagation) with the previous *error free* frame until a decoded picture without errors has been received. Since the erroneous frames are not displayed, this type of artifact is also referred to as freezing with skipping.

The flow chart in figure 4-2 describes the flow of parameters in the model and in what subsection each part can be found. Model coefficients are also given if it is being used by the algorithms.

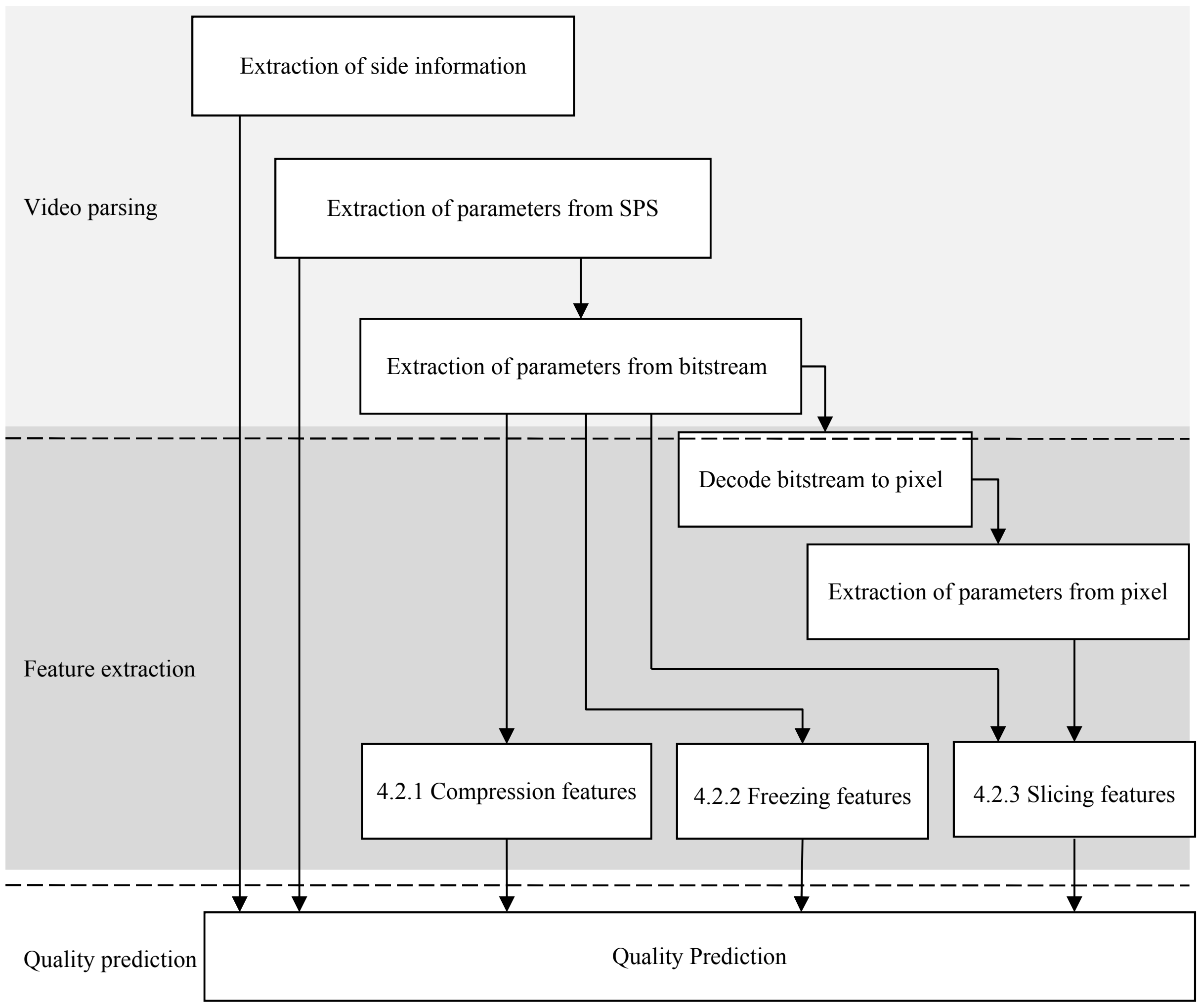

Extraction of side information
Video parsing
Extraction of parameters from SPS
Extraction of parameters from bitstream
Decode bitstream to pixel
Extraction of parameters from pixel
Feature extraction
4.2.1 Compression features
4.2.2 Freezing features
4.2.3 Slicing features
Quality prediction
Quality Prediction

Figure 4-2 Flow of parameters in the model

Generally, the algorithm consists of three stages: 1) video parsing, 2) feature extraction and 3) quality prediction as illustrated in figure 4-2. For each stage, a set of input and output parameters are marked as Data structure 1 ~ 4 in figure 4-1, and is describe as below. In the video parsing stage, the video configurations (e.g., video resolution and scanning type), the codec configurations (e.g., picture type) and the coded video data (e.g., MV, QP, DCT coefficients, EP flags indicating whether each MB is impaired or not) are parsed directly from the bit stream without full decoding.

The side information including the FR (frame rate) of the video and the PLC (Packet Loss Concealment) mode of the decoder. The side information is described by JSON as:

Data structure 1: side information

```
{“side information”: {
    “PLC mode”: <string>,
    “frame rate”: <number>
}
}
```

The parsed feature is described by JSON as:

Data structure 2: parsed feature

```
{"parsed feature":  {
    "height (H)":<number>,
    "width (W)":<number>,
    "number of pictures (T)": <number>,
    "scanning type":<string>,
    "pictures": [
                {"picture type": <string>,
                 "displaying order": <number>,
                 "MBs": [
                        {"QP": <number>,
                         "DCTs": ["coefficients of partition": <array>],
                         "Motions": [
                                    {"MV of partition": {
                                                 "vertical MV":<number>,
                                                "horizontal MV":<number>
                                     }
                                    }]
                         "EP": <Boolean>
                        }]
                }]
}
}
```

In the feature extraction stage, it is determined whether to decode the pictures and extract the pixel-layer features or not. If decoding is enabled, it outputs the decoded data as:

Data structure 3: decoded data

```
{"decoded data": {
    "impaired pictures": [
                         {"displaying order": <number>,
                          "pixels": <array>
                         }]
}
}
```

Then, all the information obtained in the previous and current stage is used to compute the necessary features. To be specific, when the decoder uses slicing PLC mode and some channel loss occur within the group of pictures between the latest and the next IDR pictures, current picture needs to be full decoded; otherwise, full decoding should be skipped. The extracted features are described as:

Data structure 4: extracted feature

```
{"extracted feature": {
    "compression": {
                    "key-factor (xc)": <number>,
                    "co-variate (zc)": <number>,
                    "frame rate (F)": <number>,
                    "resolution (R)": <number>
                   }
    "freezing":    {
                    "key-factor (xf)": <number>,
                    "co-variate (zf)": <number>
                   }
    "slicing":     {
                    "key-factor (xs)": <number>,
                    "co-variate (zs)": <number>
                   }
}
}
```

Once the features have been extracted, they are passed on to the quality model, which calculates the distortion level due to each type of video impairments, combines them and maps the result to a MOS scale. The mathematical function of the quality model is:

$$q = \frac{MOS_{\text{ub}} - MOS_{\text{lb}}}{1 + \alpha\left(a_{\text{c}} x_{\text{c}}^{b_{\text{c0}}} z_{\text{c}}^{b_{\text{c1}}} F^{b_{\text{c2}}} R^{b_{\text{c3}}} + a_{\text{f}} x_{\text{f}}^{b_{\text{f0}}} z_{\text{f}}^{b_{\text{f1}}} + a_{\text{s}} x_{\text{s}}^{b_{\text{s0}}} z_{\text{s}}^{b_{\text{s1}}}\right)^{\beta}} + MOS_{\text{lb}} \tag{4-1}$$

where output variable q is the predicted quality score; constants MOS_{ub} and MOS_{lb} are the upper bound and lower bound of MOS, i.e., 5 and 1, respectively; α, β, $\{a\}$ and $\{b\}$ are model coefficients ($a_{\text{c}} = 1$ constantly); subscripts c, f and s indicate the compression, freezing and slicing impairment respectively; variables $\{x\}$ and $\{z\}$ are model factors and also generally termed as features, which are extracted from video data. To be specific, $\{x\}$ and $\{z\}$ are the key-factor and the co-variate features associated with each type of impairment, e.g., x_{c} is the feature for compression impairment and z_{s} is the feature for slicing impairment. R and F are the number of macroblocks per picture and the frame rate, respectively.

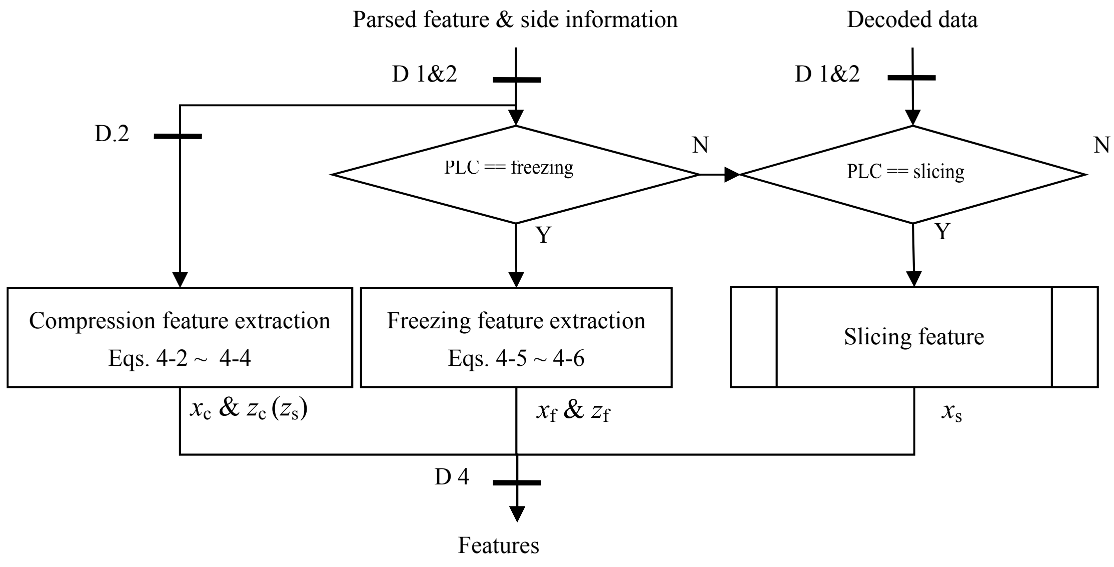


Figure 4-3 – Overview of the feature extraction

4.1 Feature extraction and aggregation

The features $\{x\}$ and $\{z\}$ in Eq. 4-1 are scalar variables for each input H.264 bit stream. The features include $\{x\}$ and $\{z\}$. The $\{x\}$ are crucial for quality prediction, while the $\{z\}$ can assist in quality prediction. Both $\{x\}$ and $\{z\}$ are grouped into three categories, associated with the three types of video impairments, i.e., compression, freezing, and slicing.

4.1.1 Compression impairment

- x_c, the feature for compression impairment, quantifies the average quantization parameter

$$x_c = 51 - \frac{1}{T}\sum_{t}\sum_{r\in t}\frac{QP_r}{|r|_t} \tag{4-2}$$

where 51 is the upper bound of QP for H.264; QP_r is the QP value of the r-th MB (macro block) within a picture; $|r|_t$ denotes the total number of unimpaired MBs in the t-th picture, and in other words, if a MB is impaired due to channel loss its QP does not contribute to x_c; T is the total number of pictures. QP, $|r|$ and T are parsed and counted during the video parsing stage.

- z_c, the feature for compression impairment, quantifies the content unpredictability (CU)

$$z_c = \frac{1}{T}\left(\sum_{t\in\{\mathrm{I}\}}\sum_{r\in t}\frac{CU_r}{20.6\times|r|_t}+\sum_{t\in\{\mathrm{P}\}}\sum_{r\in t}\frac{CU_r}{3.52\times|r|_t}+\sum_{t\in\{\mathrm{B}\}}\sum_{r\in t}\frac{CU_r}{|r|_t}\right) \tag{4-3}$$

where $t\in\{\mathrm{I}\}$, $t\in\{\mathrm{P}\}$ and $t\in\{\mathrm{B}\}$ represent I picture, P picture, and B picture, respectively; CU_r is the content unpredictability value of the r-th MB. For each macro block, the CU is defined as the variance of pixels, which is theoretically equal to the sum of the squared de-quantized DCT coefficients minus the squared sum of de-quantized DC coefficients. That is, IDCT is not necessary for calculating CU. Given the QP and quantized DCT coefficients which are parsed from bit stream directly, the de-quantized DCT coefficients can be estimated in sufficient precision. In summary, *CU* is calculated as:

$$CU_r = \frac{\left(0.625 \times 2^{QP_r/6}\right)^2}{256}\left[\sum_{l\subset r}\sum_{k\in l} DCT_{k,l}^2 - \left(\sum_{l\subset r}\frac{DCT_{0,l}}{|l|_r}\right)^2\right] \tag{4-4}$$

where the constant denominator 256 is the total number of pixels in a MB for H.264; $0.625 \times 2^{QP_r/6}$ is the approximated formula of quantization steps with respect to QP value for H.264 (this item could be omitted if de-quantized DCT coefficients are used in Eq. 4-3); l indexes the partitions in the r-th MB and $|\ l\ |_r$ denotes the total number of partitions therein, e.g., 16 partitions for 4×4 transform scheme and 4 partitions for 8×8 transform scheme; non-negative k indexes the quantized DCT coefficients in the l-th partition of the r-th MB, and $DCT_{0,l}$ implies the quantized DC coefficient in the l-th partition. Since even for the same picture content CU values may vary with the picture types, the constant weights (i.e., 20.6, 3.52, and 1) in Eq. 4-3 are used to align CUs of each type of pictures to a same numerical scale.

4.1.2 Freezing impairment

- x_f, the feature for freezing impairment, quantifies the freezing duration.

$$x_\mathrm{f} = \frac{1}{T}\sum_\tau FD_\tau^{0.9} \tag{4-5}$$

where T is the total number of pictures in the input video clip, FD_τ represents the duration of the τ-th pause. If the last pause lasts until the end of the video clip and is shorter than 2 second, it is not taken into account for $\{FD_\tau\}$.

- z_f, the feature for freezing impairment, quantifies the motion homogeneity.

$$z_\mathrm{f} = \frac{1}{P}\sum_\tau \max\{IH_\tau, RH_\tau\} \tag{4-6}$$

where P is the total number of the visual pauses. *IH* quantifies the isotropic homogeneity while *RH* specifies the radial homogeneity. For each picture, *IH* is defined as the magnitude of the vector mean of all MVs in the picture as:

$$IH_\tau = \frac{1}{H\cdot W}\sqrt{\left(\sum_{r\in\tau}\sum_{l\subset r} MV_{\mathrm{h},l,r}\cdot A_{l,r}\right)^2 + \left(\sum_{r\in\tau}\sum_{l\subset r} MV_{\mathrm{v},l,r}\cdot A_{l,r}\right)^2} \tag{4-7}$$

where r indexes the MBs in the τ-th picture and l indexes the partitions in the r-th MB; $MV_{\mathrm{h},l,r}$ and $MV_{\mathrm{v},l,r}$ and $A_{l,r}$ denote the horizontal MV, vertical MV and area (number of pixels) of the l-th partition in the r-th MB respectively; all the *MV*s were clipped into the range of [−128, 128] and multiplied with the frame rate of the video so that it only depends on the content motion but not the video frame rate; constants H and W are the height and width of the picture. *IH* is greater when a big foreground in the scene is translating or the camera is panning.

RH is more complicated to compute. Firstly, the difference between the sum of horizontal MVs in the left half picture and those in the right half picture, and the difference between the sum of vertical MVs in the top half picture and those in the bottom half picture are both calculated. Secondly, the two difference values are both normalized by the total number of MB in a picture, i.e., $H\cdot W$, to form a 2D vector. Thirdly, *RH* is set as the magnitude of the vector.

$$\mathrm{RH}_\tau = \frac{1}{HW}\sqrt{\left|\sum_{r\in\tau_\mathrm{L}}\sum_{l\subset r} MV_{\mathrm{h},l.r}\,A_{l,r} - \sum_{r\in\tau_\mathrm{R}}\sum_{l\subset r} MV_{\mathrm{h},l.r}\,A_{l.r}\right|^2 + \left|\sum_{r\in\tau_\mathrm{T}}\sum_{l\subset r} MV_{\mathrm{v},l.r}\,A_{l,r} - \sum_{r\in\tau_\mathrm{B}}\sum_{l\subset r} MV_{\mathrm{v},l.r}\,A_{l,r}\right|^2} \quad (4\text{-}8)$$

where τ_L, τ_R, τ_T and τ_B represent the left, right, top and bottom half plane of the τ-th picture respectively. *RH* is greater if the camera is zooming, since *RH* is cumulated over the radial symmetric motion vectors. IH_τ and RH_τ are respectively the *IH* and the *RH* of the intact inter-predicted picture (i.e., P or B picture) just before the τ-the visual pause. For the τ-th pause, the greater value between IH_τ and RH_τ is selected. z_f is defined as the average greater values over the total of *P* pauses.

4.1.3 Slicing impairment

- z_s, the feature of slicing impairment. z_s is same as z_c, referring to Eqs 4-3 and 4-4.

- x_s, the feature of slicing impairment, quantifies the visible error rate.

First of all, the input data of the overall x_s extraction (as captured by figure 4-4) are defined as:

Data structure 5: inputs of x_s extraction

```
{"inputs of xs extraction": {
   "impaired pictures": [
      {"MBs": [
              {"EP": <Boolean>,
               "motions": ["MV of partition": <number>],
               "pixels": <array>
              }]
       }]
}
}
```

wherein, after feature extraction and aggregation, the data below are passed on to Eq. 4-9.

Data structure 6: inputs of Eq. 4-9

```
{"inputs of Eq. 4-9": {
    "impaired pictures": [
       {"impaired MBs": [
                      {"MF": <Boolean>,
                       "MA": <number>,
                       "AE": <number>,
                       "TX": <number>
                      }]
       }]
}
}
```

$$x_\mathrm{s} = \sum\nolimits_t \left[\sum_{\{r|EP_{r,t}=1\}} \frac{MF_{r,t}\left(MA_{r,t}+AE_{r,t}\right)^{1.5}}{\left(1+TX_{r,t}\right)\cdot R}\right]^{0.7} \Big/ T \quad (4\text{-}9)$$

x_s is calculated by the inner summation over MBs (spatially), the outer summation over pictures (temporally), where *r* indexes the MB in a frame, *t* indexes the picture in a video clip, *R* is the total number of MBs in a picture and *T* is the total number of pictures in the input video clip. *EP* and *MF* are MB-wise Boolean features, respectively indicating whether the MB is propagated by packet loss or not, and whether the MB would be recovered by error concealment due to its motion factor or not. *MA*, *AE* and *TX* quantify the suspected mosaic artifacts, the suspected false edge artifacts, and the texture strength of the MB respectively. EP flag is tagged to each MB during the video parsing stage. *EP* is true if the MB is lost or the MB (directly or indirectly) uses lost MBs for prediction. Other quantities are calculated as descriptions (a ~ d) below. For an illustration, see figure 4.5.

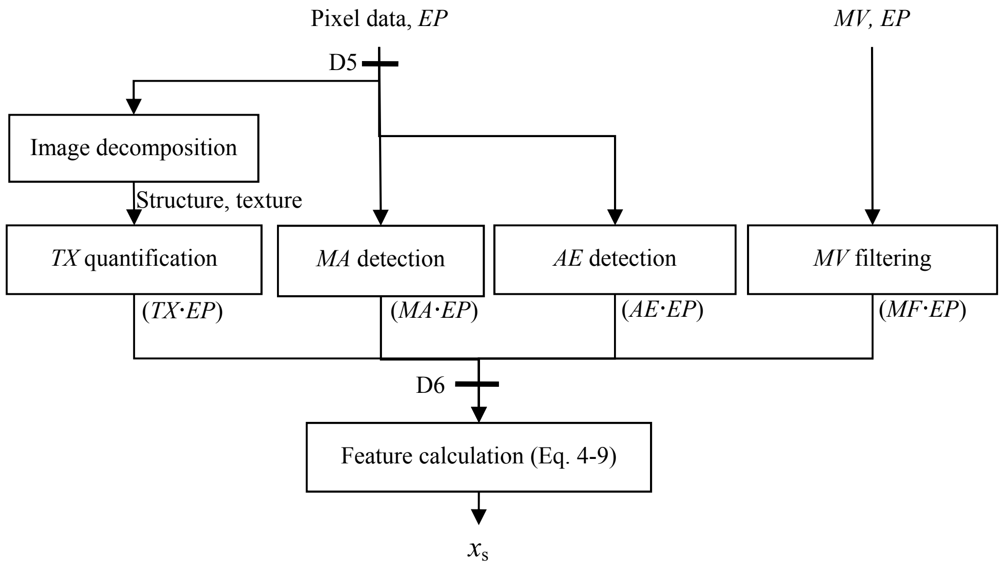


Figure 4-5 – Slicing feature extraction

a) *MV* filtering

MV filtering is suggested when the decoder uses the error concealment method of copying co-located pixels. In such case, the MBs containing still content are able to be recovered finely when they are impaired. *MV* filtering sets the *MF* of a MB false if and only if all the received MBs within its 26 neighbours (in 3D spatial-temporal space) have zero MVs, otherwise sets the *MF* true.

b) Mosaic artifacts (*MA*) detection

The mosaic artifact *MA* is set to be true when unsmooth vertical gradients along the regular MB grids are detected as Eq. 4-10. The second-order gradient sum for each vertically adjacent MB pair is calculated as Eq. 4-11 or 4-12. Each MB corresponds to an upper sum ∇_{u}^2 and a lower gradient sum ∇_{l}^2. The smaller one is selected as the mosaic artifact occurs on the MB as:

$$MA = \min\{|\nabla_{\mathrm{u}}^2|, |\nabla_{\mathrm{l}}^2|\} \qquad (4\text{-}10)$$

Denote a MB pair by the upper MB consisting of pixels $\{p_{i,j} \mid i = 1,2, \ldots, 16 \text{ and } j = 1, 2, \ldots, 16\}$ and the lower MB consisting of pixels $\{p_{i,j} \mid i = 17,18, \ldots, 32 \text{ and } j = 1, 2, \ldots, 16\}$, where *i* and *j* index the row and column of pixels, respectively.

For progressive scanning video, ∇_{l}^2 of the upper MB and ∇_{u}^2 of the lower MB are both calculated as:

$$\nabla_{\mathrm{progressive}}^2 = \min\left\{\left|\sum_{j=1}^{16}(p_{18,j} + p_{16,j} - 2p_{17,j})\right|, \left|\sum_{j=1}^{16}(p_{17,j} + p_{15,j} - 2p_{16,j})\right|\right\} \qquad (4\text{-}11)$$

For interlaced scanning video, ∇_{l}^2 of the upper MB and ∇_{u}^2 of the lower MB are both calculated as:

$$\nabla^2_{\text{Interlaced}} = \min\left\{\left|\sum_{j=1}^{16}(p_{18,j}+p_{14,j}-2p_{16,j})\right|, \left|\sum_{j=1}^{16}(p_{19,j}+p_{15,j}-2p_{17,j})\right|, \left|\sum_{j=1}^{16}(p_{18,j}+p_{16,j}-2p_{17,j})\right|, \left|\sum_{j=1}^{16}(p_{17,j}+p_{15,j}-2p_{16,j})\right|\right\} \qquad (4\text{-}12)$$

where the minimum of the four values is selected and thus the discontinuity between the interlaced fields is tolerated to some extent.

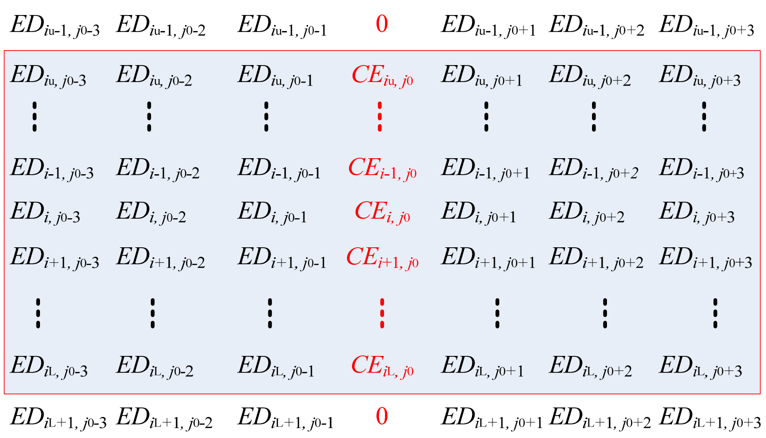
Only the *MA*s which involve impaired MBs need to be determined, while other *MA*s can be omitted.

c) Artificial edges (*AE*) detection

Artificial edges detection is supplementary to Mosaic detection, since any unnatural edges outside regular MB boundaries are counted. Firstly, the vertical edges are detected as (the horizontal artificial edges may be detected in a similar way):

$$ED_{i,j} = \left|(p_{i-1,j}-p_{i-1,j+1}) + 2(p_{i,j}-p_{i,j+1}) + (p_{i+1,j}-p_{i+1,j+1})\right|/4 \qquad (4\text{-}13)$$

where i and j index the row and column of pixels, respectively; $p_{i,j}$ and $ED_{i,j}$ are the pixel value and the edge strength corresponding to pixel (i, j). The edge strengths are set to 0 along the image boundary where $p_{i-1,j}$ or $p_{i+1,j}$ is out of image boundary.



$ED_{i_u-1,\,j_0-3}$	$ED_{i_u-1,\,j_0-2}$	$ED_{i_u-1,\,j_0-1}$	0	$ED_{i_u-1,\,j_0+1}$	$ED_{i_u-1,\,j_0+2}$	$ED_{i_u-1,\,j_0+3}$
$ED_{i_u,\,j_0-3}$	$ED_{i_u,\,j_0-2}$	$ED_{i_u,\,j_0-1}$	$CE_{i_u,\,j_0}$	$ED_{i_u,\,j_0+1}$	$ED_{i_u,\,j_0+2}$	$ED_{i_u,\,j_0+3}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$ED_{i-1,\,j_0-3}$	$ED_{i-1,\,j_0-2}$	$ED_{i-1,\,j_0-1}$	$CE_{i-1,\,j_0}$	$ED_{i-1,\,j_0+1}$	$ED_{i-1,\,j_0+2}$	$ED_{i-1,\,j_0+3}$
$ED_{i,\,j_0-3}$	$ED_{i,\,j_0-2}$	$ED_{i,\,j_0-1}$	$CE_{i,\,j_0}$	$ED_{i,\,j_0+1}$	$ED_{i,\,j_0+2}$	$ED_{i,\,j_0+3}$
$ED_{i+1,\,j_0-3}$	$ED_{i+1,\,j_0-2}$	$ED_{i+1,\,j_0-1}$	$CE_{i+1,\,j_0}$	$ED_{i+1,\,j_0+1}$	$ED_{i+1,\,j_0+2}$	$ED_{i+1,\,j_0+3}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$ED_{i_L,\,j_0-3}$	$ED_{i_L,\,j_0-2}$	$ED_{i_L,\,j_0-1}$	$CE_{i_L,\,j_0}$	$ED_{i_L,\,j_0+1}$	$ED_{i_L,\,j_0+2}$	$ED_{i_L,\,j_0+3}$
$ED_{i_L+1,\,j_0-3}$	$ED_{i_L+1,\,j_0-2}$	$ED_{i_L+1,\,j_0-1}$	0	$ED_{i_L+1,\,j_0+1}$	$ED_{i_L+1,\,j_0+2}$	$ED_{i_L+1,\,j_0+3}$

Figure 4-6 Neighbourhood for determining the saliency of continuous edge (i_{u}, i_{l}, j)

Secondly, the continuous edges are grouped as:

$$CE_{i,j} = \begin{cases} ED_{i,j}, & \forall\ i_{\mathrm{u}} \text{ and } i_{\mathrm{l}}, \quad \text{s.t. } i_{\mathrm{l}} - i_{\mathrm{u}} \geq 5 \text{ and } ED_{i,j} \geq 12 \text{ for all } i_{\mathrm{u}} \leq i \leq i_{\mathrm{l}} \\ 0\ , & \text{else} \end{cases} \qquad (4\text{-}14)$$

where $CE_{i,j}$ is kept non-zero only if an edge stronger than 11 gray-levels and longer than 5 pixels goes through pixel (i, j). Consequently, each continuous edge is denoted by a 3-tuple set (i_{u}, i_{l}, j), which consists of (i_{l} - i_{u}+1) non-zero elements$\{CE_{i,j} | i_{\mathrm{u}} \leq i \leq i_{\mathrm{l}}\}$.

Thirdly, the continuous edges are filtered and only salient edges survive. For each continuous edge (i_{u}, i_{l}, j_0), its (i_{l}−i_{u}+1)×7 neighbours are inspected as shown in figure 4-6. Continuous edge (i_{u}, i_{l}, j_0) is marked salient only if:

$$\frac{\sum_{i_\mathrm{u} \le i \le i_\mathrm{l}} CE_{i,j_0}}{i_\mathrm{l} - i_\mathrm{u} + 1} \ge \min\{\delta + 5, \delta/0.6\}$$

$$\delta = \max\left\{ \frac{\sum_{i_\mathrm{u} \le i \le i_\mathrm{l}, ED_{i,j} \ge 5} ED_{i,j}}{|i|_{i_\mathrm{u} \le i \le i_\mathrm{l}, ED_{i,j} \ge 5}} \middle| j_0 - 3 \le j \le j_0 - 3, j \ne j_0 \right\} \tag{4-15}$$

where $|i|_{i_\mathrm{u} \le i \le i_\mathrm{l}, E_{i,j} \ge 5}$ denotes the total number of pixels between (i_u, j) and (i_l, j) satisfying $ED_{i,j} \ge 5$. Condition 4-15 being true suggests that the current continuous edge (i_u, i_l, j_0) is stronger than any parallel edges in surrounding and thus looks salient.

Fourthly, any isolated edges are excluded further. For each salient edge (i_u, i_l, j), V^- denotes the total number of pixels between (i_u, j−16) and (i_l, j−16) which belongs to any other salient edge, and equivalently V^+ denotes the total number of pixels between (i_u, j+16) and (i_l, j+16) which belongs to any another salient edge. The salient edge (i_u, i_l, j) is not excluded only if:

$$i_\mathrm{l} - i_\mathrm{u} + 1 \le 0.5 \cdot \max\{V^-, V^+\} \tag{4-16}$$

Condition 4-16 being true suggests that a pair of parallel salient edges occurs just around a (potential propagated) MB boundary, and thus more likely indicates artificial edges rather than natural edges.

Finally, *AE* of a MB is the total number of the remained salient edges located in/across the MB. Only the *AE*s which involve impaired MBs need to be determined, while other *AE*s can be omitted.

Algorithm : Artificial edges (*AE*) estimation

```
Input:  picture $\boldsymbol{p} \in \mathbb{Z}^{H\times W}$
Output: texture mask $AE \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}}$

//1. Vertical edge detection
for each pixel $p(i,j)$ ($2 \leqslant i \leqslant H-1$, $1 \leqslant j \leqslant W-1$)
    $ED(i,j) = |[p(i-1,j) - p(i-1,j+1)] + 2[p(i,j) - p(i,j+1)] + [p(i+1,j) - p(i+1,j+1)]|/4$
//2. Continuous edge grouping
Initial $CES$ as a null 3-tuple set, $CE$ as a $H\times W$ integral zero array
for each pixel $p(i,j)$ ($2 \leqslant i \leqslant H-1$, $1 \leqslant j \leqslant W-1$)
    if $ED(i,j) \geq 12$
        $u = 0, l = 0$
        while $ED(i+u,j) \geq 12$
            $u++$
        while $ED(i-l,j) \geq 12$
            $l++$
        If $u + l \geq 5$
            $CE(i,j) = ED(i,j)$
            If $(i+u, i-l, j) \notin CES$
                Add $(i+u, i-l, j)$ into set $CES$
//3. Salient edge filtering
for each 3-tuple $(i_u, i_l, j)$ in $CES$
    for each column j+j'-4 ($1 \leqslant j' \leqslant 7, j' \neq 3$)
        $s$=0, $S$=0
        for each neighbour p(i',j+j'-4) ($i_l \leqslant i' \leqslant i_u$)
            if $ED(i', j+j'-4) \geq 5$
                $S \mathrel{+}= ED(i', j+j'-4)$
                $s++$
        $\delta(j') = S/s$
    $\delta = \max\{\delta(1:7)\}$
    $S$ = 0
    for pixel $p(i,j)$ ($i_l \leqslant i \leqslant i_u$)
        $S \mathrel{+}= p(i,j)$
    $S \mathrel{/}= i_u - i_l + 1$
    if $S < \delta$
        remove $(i_u, i_l, j)$ from $CES$
//4. Excluding isolated edges
for each remained 3-tuple $(i_u, i_l, j)$ in $CES$
    if $i_u - i_l + 1 > 16$
        $V^- = 0, V^+ = 0$
        for pixel $p$(i,j-16), ($i_l \leqslant i \leqslant i_u$)
            if $\exists(i_u', i_l', j-16)$ satisfying $i_l' \leq i \leq i_u'$
                $V^-++$
        for pixel $p$(i,j+16), ($i_l \leqslant i \leqslant i_u$)
            if $\exists(i_u', i_l', j+16)$ satisfying $i_l' \leq i \leq i_u'$
                $V^+++$
        if $i_u - i_l + 1 > 0.5\max\{V^-, V^+\}$
            remove $(i_u, i_l, j)$ from $CES$
//5. Artificial edge assignment
Initialize AE as a $(H/16)\times(W/16)$ integral zero array
for each 16×16 block (16$i$-15:16$i$, 16$j$-15:16$j$)
    for each pixel $p$(i',j') in the current block
        if $\exists(i_u, i_l, j')$  satisfying $i_l \leq i' \leq i_u$
            $AE(i,j)++$
return $AE$
```

d) Texture masking estimation

First of all, each picture is decomposed into the structure and texture components. It includes the steps below.

Firstly each picture is decomposed to a structure and a texture component by 7×7 bilateral filter. The filtered picture is deemed as the structure component, while the difference between the original picture and the filtered structure is deemed as the texture component.

Secondly, an edge map is labelled where the Sobel filtering response on the structure component is above a threshold 150. Before filtering, the structure component is downsampled at 1:16 so that each "pixel" in the downsampled structure component corresponds to a MB.

Thirdly, texture strength is calculated from the MBs' variance in the texture component. The texture strength is set to 0 if the corresponding pixel in the edge map indicates an edge.

Only the *TX*s which involve impaired MBs need to be determined, while other *TX*s are set to 0.

Algorithm : Texture mask (*TX*) estimation

```
Input:  picture $\boldsymbol{p} \in \mathbb{Z}^{H\times W}$, $EP \in \mathbb{B}^{\frac{H}{16}\times\frac{W}{16}}$
Output: texture mask $TX \in \mathbb{R}^{\frac{H}{16}\times\frac{W}{16}}$

Initialize $TX$ as a (H/16)×($W$/16) float zero array
Initialize $structure$ as an $H$×$W$ float zero array
for each pixel $p(i,j)$ ($4 \leqslant i \leqslant H-3$, $4 \leqslant j \leqslant W-3$)
    $k = 0$
    for each neighbour $p(i',j')$ ($i-3 \leqslant i' \leqslant i+3$, $j-3 \leqslant j' \leqslant j+3$)
        $structure(i,j) += p(i',j')e^{-|p(i,j)-p(i',j')|-\sqrt{(i-i')^2+(j-j')^2}}$        //bilateral filter
        $k += e^{-|p(i,j)-p(i',j')|-\sqrt{(i-i')^2+(j-j')^2}}$
    $structure(i,j)$ /= $k$
$texture$ = $\boldsymbol{p}$ - $structure$                          //image decomposition
for each unoverlapped 16×16 block $structure$(16$i$-15:16$i$, 16$j$-15:16$j$)
    $ST(i,j)$ = mean($structure$(16$i$-15:16$i$, 16$j$-15:16$j$))  //downsampling
//Initialize $ED$ as a (H/16)×($W$/16) float array full of elements 150
for each $STRUCTURE(i,j)$ ($2 \leqslant i' \leqslant H/16-1$, $2 \leqslant j' \leqslant W/16-1$)
    if $EP(i,j)$== TRUE
        $ED(i,j)$ = |$ST(i-1,j+1)$+2$ST(i,j+1)$+$ST(i+1,j+1)$
                    -$ST(i-1,j-1)$-2$ST(i,j-1)$-$ST(i+1,j-1)$| +
                    |$ST(i+1,j-1)$+2$ST(i+1,j)$+$ST(i+1,j+1)$
                    -$ST(i-1,j-1)$-2$ST(i-1,j)$-$ST(i-1,j+1)$| //Sobel filter
for each unoverlapped 16×16 block $texture$(16$i$-15:16$i$, 16$j$-15:16$j$)
    if $ED(i,j)$ < 150   //impaired non-edge region
        $TX(i,j)$ = variance($texture$(16$i$-15:16$i$, 16$j$-15:16$j$))
return $TX$
```

4.2 Coefficient setting

The quality model supports a single set of coefficients (α, β, $\{a\}$ and $\{b\}$) for all HBR videos (i.e., for all resolutions, frame rates, and progressive/interlaced formats). The model employed only one set of coefficients when it won the phase of model competition. However, during the phase of model optimization, multiple sets of coefficients can achieve better performance values. In such a manner, three sets of coefficients are used for SD, HD720, and HD1080 videos, respectively. After optimization over all the thirteen ITU-T SG 12 databases, the coefficient setting are recommended as Table 4-1 and Table 4-2.

Table 4-1 Single set of coefficients

a_c	$\ln a_f$	$\ln a_s$	b_{c0}	b_{c1}	b_{c2}	b_{c3}	b_{f0}	b_{f1}	b_{s0}	b_{s1}	$\ln\alpha$	β
1	-1.42	-3.84	-1.58	-.173	1.03	-.337	1.01	.135	.779	-.122	3.13	.967

Table 4-2 Multiple sets of coefficients adaptive to video resolution

	a_c	$\ln a_f$	$\ln a_s$	b_{c0}	b_{c1}	b_{c2}	b_{c3}	b_{f0}	b_{f1}	b_{s0}	b_{s1}	$\ln\alpha$	β
SD	1	-4.45	-6.83	-2.51	-.262	0	0	1.18	.137	.880	-.244	4.82	.734
720	1	-2.03	-4.99	-1.35	-.144	0	0	.985	.087	.842	-.107	4.60	1.27
1080	1	-1.31	-3.50	-1.21	-.147	0	0	.908	.118	.749	-.135	3.32	1.15

5 Assumptions

It is assumed that the user of this Recommendation knows how to extract an MPEG-TS/RTP packed video bitstream encapsulated in the libPcap format, as well as extracting the video elementary stream from the MPEG-TS packed video bitstream, while keeping track of lost pictures.

It is further assumed that the user of this Recommendation has good knowledge about the H.264/AVC video codec defined in the H.264 Recommendation, and that the user knows how to extract basic parameters and bitstream information from the video bitstream.

6 Model compliance

In order to verify compliance with Recommendation ITU-T P.1202.2 model mode 1 and mode 2 a set of 6 test sequences with a set of corresponding test vectors respectively are provided with Recommendation ITU-T P.1202.2.

The following test files have been provided with this Recommendation for verifying compliance with the ITU-T P.1202.2 (HR) model:

Test vectors for mode 1:

P1202_HR_Mode1_TV01.pre
P1202_HR_Mode1_TV02.pre
P1202_HR_Mode1_TV03.pre
P1202_HR_Mode1_TV04.pre
P1202_HR_Mode1_TV05.pre
P1202_HR_Mode1_TV06.pre

P1202_HR_Mode1_TV01.pcap
P1202_HR_Mode1_TV02. pcap
P1202_HR_Mode1_TV03. pcap
P1202_HR_Mode1_TV04. pcap
P1202_HR_Mode1_TV05. pcap
P1202_HR_Mode1_TV06. Pcap

Test vectors for mode 2:

P1202_HR_Mode2_TV01.pre
P1202_HR_Mode2_TV02.pre
P1202_HR_Mode2_TV03.pre
P1202_HR_Mode2_TV04.pre
P1202_HR_Mode2_TV05.pre
P1202_HR_Mode2_TV06.pre

P1202_HR_Mode2_TV01.pcap
P1202_HR_Mode2_TV02. pcap
P1202_HR_Mode2_TV03. pcap
P1202_HR_Mode2_TV04. pcap
P1202_HR_Mode2_TV05. pcap
P1202_HR_Mode2_TV06. Pcap

6.1 Test vector for quality estimation model of mode 1

Table 6-1 - Test vector for no packet losses

pcap	**TV01**	**TV02**
f_video_qp	21.622	32.334
f_video_content_complexity	40.376091	37.534088
d_compression_quality_value	4.431	4.028

Table 6-2 - Test vectors for packet losses - slicing plc

pcap	**TV03**	**TV04**
f_video_qp	21.622	24.560242
f_video_content_complexity	40.376091	51.633858
d_compression_quality_value	4.431	4.409
d_slicing_artifact_value	4.682360726	4.890516485
d_combined_quality_value	2.412	2.217

Table 6-3 - Test vectors for packet losses - freezing plc

pcap	**TV05**	**TV06**
f_video_qp	21.622	24.973948
f_video_content_complexity	40.376091	52.208214
d_compression_quality_value	4.431	4.404
f_freezing_ratio	0.422	0.056
f_fps	50	50
i_total_num_freezing_frames	211	28
i_total_num_frames	500	500
d_MV	2.990238095	0.656666667
d_freezing_artifact_value	3.068674255	1.278976309
d_combined_quality_value	1.878	3.583

6.2 Test vector for quality estimation model of mode 2

Table 6-4 - Test vectors for no packet losses

pcap	**TV01**	**TV02**
F (frame rate)	25	50
R (resolution)	8160	3600
x_c (quantization)	27.4206	31.4551
z_c (unpredictablility)	3.0695	12.601

Table 6-5 - Test vectors for packet losses - slicing plc

pcap	**TV03**	**TV04**
F (frame rate)	25	50
R (resolution)	8160	3600
x_c (quantization)	29.7188	25.599
z_c (unpredictablility)	2.8161	16.1776
x_s (error)	5.5566	8.4363
z_s (unpredictablility)	2.8161	16.1776

Table 6-6 - Test vectors for packet losses - freezing plc

pcap	**TV03**	**TV04**
F (frame rate)	25	50
R (resolution)	8160	3600
x_c (quantization)	24.0432	25.2845
z_c (unpredictablility)	4.1123	16.828
x_f (duration)	0.7455	0.5053
z_f (motion)	19.733	2.8358

The .pcap files are H.264/AVC video bitstream files packed in MPEG-TS/RTP/UDP/IP encapsulated in the libpcap dump format. The .pre contains side information that is used by the P.1202-HR model.

7 Bibliography

[Zhang, et.al., 2013] F. Zhang, W. Lin, Z. Chen, and K. N. Ngan, “Additive log-logistic model for networked video quality assessment.” IEEE Trans. Image Process., Vol. 22, No. 4, pp. 1536 – 1547, Apr. 2013.